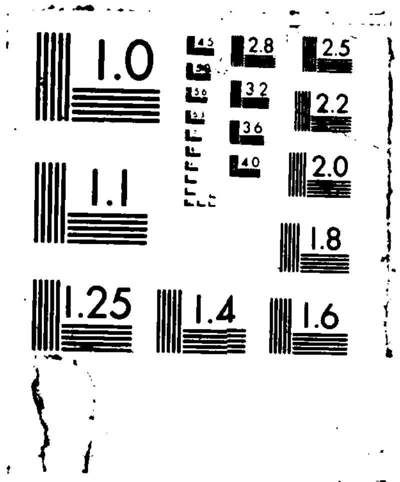
AD-A189 042   KNOWLEDGE RETRIEVAL AS SPECIALIZED INFERENCE(U)   1/2
ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE   A M FRISCH
MAY 87 TR-214 N00014-80-C-0197

UNCLASSIFIED   F/G 12/9   NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

Knowledge Retrieval as Specialized Inference

Alan Mark Frisch
Department of Computer Science
The University of Rochester
Rochester, NY 14627

TR 214
May 1987

rochester

Department of Computer Science
University of Rochester
Rochester, New York 14627

4

Knowledge Retrieval as Specialized Inference

Alan Mark Frisch
Department of Computer Science
The University of Rochester
Rochester, NY 14627

TR 214
May 1987

Author's current address:
Department of Computer Science
University of Illinois
1304 West Springfield Avenue
Urbana, Illinois 61801.

With slight modification, this report reproduces a dissertation submitted in partial fulfillment of the requirements for the degree Doctor of Philosophy in Computer Science at the University of Rochester, supervised by James F. Allen.

At the University of Rochester this research was supported in part by the Office of Naval Research under grant N00014-80-C-0197 and by the National Science Foundation under grant DCR-8351665. At the University of Sussex it was supported in part by both the Science and Engineering Research Council and the Alvey Directorate under grant SERC GR/D/16062.

[illegible]

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR 214	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Knowledge Retrieval as Specialized Inference		5. TYPE OF REPORT & PERIOD COVERED technical report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Alan M. Frisch		8. CONTRACT OR GRANT NUMBER(s) DCR-8351665
9. PERFORMING ORGANIZATION NAME AND ADDRESS Dept. of Computer Science University of Rochester Rochester, NY 14627		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS DARPA/1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE May 1987
		13. NUMBER OF PAGES 103
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. *(of this report)* unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

knowledge retrieval
limited inference
semantic networks
knowledge representation
logic

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Artificial intelligence reasoning systems commonly contain a large corpus of declarative knowledge, called a knowledge base (KB), and provide facilities with which the system's components can retrieve this knowledge. This thesis sets out to study the very nature of retrieval. Formal specifications that capture certain informal intuitions about retrieval are developed, studied, and implemented by retrieval algorithms.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. ABSTRACT (Continued)

Consistent with the necessity for fast retrieval is the guiding intuition that a retriever is, at least in simple cases, a pattern matcher, though in more complex cases it may perform selected inferences such as property inheritance. Seemingly at odds with this intuition, this thesis views the entire process of retrieval as a form of inference and hence the KB as a representation, not merely a data structure. A retriever makes a limited attempt to prove that a queried sentence is a logical consequence of the KB. When constrained by the *no-chaining-restriction*, inference becomes indistinguishable from pattern matching. Imagining the KB divided into quanta, a retriever that respects this restriction cannot combine two quanta in order to derive a third.

The techniques of model theory ~~are adapted to~~ build non-procedural specifications of retrievability relations, which determine what sentences are retrievable from what KBs. Model-theoretic specifications are presented for four retrievers, each extending the capabilities of the previous one. Each is accompanied by a rigorous investigation into its properties and a presentation of an efficient, terminating algorithm that provably meets the specification.

For my parents,

Frances and Hubert Frisch

Curriculum Vitae

Alan Frisch was born in Newark, New Jersey on June 19, 1954. Following his 1972 graduation from West Orange High School he fled New Jersey for Carnegie-Mellon University where he majored in mathematics but mostly studied computer science and psychology. During his undergraduate career he acquired a mild, though increasing addiction to artificial intelligence, taking many courses in the subject and working for two years on computer vision in the Artificial Intelligence Laboratory. He also spent a summer at Argonne National Laboratory as a graphics programmer. In December, 1976 he completed his studies at Carnegie-Mellon and was awarded a B.S. in Mathematics.

He spent the next year and a half as a development programmer at IBM in Poughkeepsie, New York, but maintained his sanity by taking graduate computer science courses given by Syracuse University.

His craving for A.I. grew and in the autumn of 1978 he began a long, active and long career as a graduate student in computer science at the University of Rochester. He was granted an M.S. in Computer Science in 1982. During his tenure at Rochester he served as a teaching assistant, research assistant, and instructor. His research included work on natural language processing, knowledge representation and logic programming, and he was a principal contributor to the ARGOT natural language system and the HORNE logic programming system.

In August, 1983 the veteran graduate student moved to Brighton, England for a one year appointment at the University of Sussex as a Lecturer in Artificial Intelligence (i.e., a British Assistant Professor). Subsequently, the Science and Engineering Research Council and the Alvey Directorate awarded him a grant to study inference in sorted logic and he stayed in the Cognitive Studies Programme at Sussex for an additional two years.

By 1986 he learned to direct his habitual A.I. activities towards productive ends and in August of that year he successfully defended his Ph.D. thesis at Rochester.

Alan Frisch is now back in the colonies as an Assistant Professor with the Supercomputing Illini in the Computer Science Department of the University of Illinois at Urbana-Champaign. He resides in Urbana with his English wife and international beer can collection.

Acknowledgements

Over six years of work on knowledge retrieval has provided me the time, opportunity and good fortune to encounter and be influenced by many people. I thank all those who have encouraged me, who have listened critically to my ideas, and who have shared their best ideas with me.

I thank my thesis committee, James Allen, Chris Brown, Henry Kyburg, and Haim Mendelson for their time, their comments, and their patience in dealing with the difficulties of arranging a defense for a student 4000 miles away. I am especially grateful to Henry Kyburg for his detailed comments on my manuscript and for helping to make the terminology of this thesis as standard and as consistent as possible.

James Allen, my thesis advisor, originally interested me in knowledge retrieval and steered me in the right direction for many years. For four years I had the privilege of collaborating with him on a near-daily basis and from this apprenticeship I came to understand the ins and outs of research. For this he has my deepest gratitude; but mostly I thank him for making my early graduate years fun.

If anything of mathematical beauty lies between these covers, then I must credit Pat Hayes. If not, I thank him anyway; without the countless hours he spent listening to, cursing at and rewriting the theorems and proofs of my early work, I would never have learned how to construct proofs, beautiful or otherwise. I also thank him for serving as an initially official and later unofficial member of my thesis committee.

Had it not been for David McAllester I would long ago have completed this thesis, or, more precisely, a vastly inferior thesis of the same name. I had almost completed the syntactic specifications of my retrievers in December 1982 when lengthy discussions with him convinced me that I could give semantic accounts of the retrievers. The discussion also provided me with initial methods to approach the task, which after three years of work resulted in the specifications contained in this thesis. I consider the quality of this work to be an outgrowth of Dave's insight and I am extremely grateful to him for sharing his ideas with me.

The reader familiar with the work of Peter Patel-Schneider will recognize the debt that I owe him. Some of the most important logical constructions used in this thesis are modifications of constructions that he has used in his own work. I thank him for sharing his best ideas with me; had he not done so, the foundation of this work would have crumbled. I can only hope that he benefits from my work as I have from his.

I am grateful to the friends and colleagues I had at the University of Rochester. Much of what I know about logic came not from textbooks but from the continual discussions that I had over several years with Andy Haas, Don Perlis, and Henry Kautz. The AI Study Group and the ARGOT research project provided a stimulating atmosphere in which to work. I am particularly grateful to the initial users of ARGOT's knowledge retriever, Mark Guiliano, Diane Litman and Marc Villain, whose patience and cooperation helped to debug the system.

I was privileged to receive constant encouragement, support, and stimulation from some of my colleagues at the University of Sussex, most notably Gerald Gazdar, Chris Mellish, and Aaron Sloman. I thank Chris and Aaron for their assistance in obtaining a grant to support my work, and Aaron for reading and criticizing almost everything I wrote.

In various ways I have been aided by many people in the field and, in particular, I thank Ron Brachman, Tony Cohn, John Kender, Hector Levesque, Allen Newell, Alan Robinson, and Christoph Walther.

I'm sure that every Computer Science Ph.D. graduate from Rochester understands the deep appreciation that I have for Jill Orioli Forster. It might have been possible to complete this thesis without her, but it would have been a hell of a lot harder.

For her loyal love and support, for the boundless energy she invested in assisting me on many long nights of work, for drawing all the figures in this thesis, for correcting, recorrecting, and re-recorrecting my spelling, for proofreading every page of this thesis, and for helping me to omit many needless words, I cannot even begin to express the gratitude I have for my wife, Frances Evelyn. I can only hope for the privilege of reciprocating during the completion of her thesis.

Abstract

Artificial Intelligence reasoning systems commonly contain a large corpus of declarative knowledge, called a knowledge base (KB), and provide facilities with which the system's components can retrieve this knowledge. This thesis sets out to study the very nature of retrieval. Formal specifications that capture certain informal intuitions about retrieval are developed, studied, and implemented by retrieval algorithms.

Consistent with the necessity for fast retrieval is the guiding intuition that a retriever is, at least in simple cases, a pattern matcher, though in more complex cases it may perform selected inferences such as property inheritance.

Seemingly at odds with this intuition, this thesis views the entire process of retrieval as a form of inference and hence the KB as a representation, not merely a data structure. A retriever makes a limited attempt to prove that a queried sentence is a logical consequence of the KB. When constrained by the *no-chaining restriction,* inference becomes indistinguishable from pattern-matching. Imagining the KB divided into quanta, a retriever that respects this restriction cannot combine two quanta in order to derive a third.

The techniques of model theory are adapted to build non-procedural specifications of retrievability relations, which determine what sentences are retrievable from what KB's. Model-theoretic specifications are presented for four retrievers, each extending the capabilities of the previous one. Each is accompanied by a rigorous investigation into its properties, and a presentation of an efficient, terminating algorithm that provably meets the specification.

The first retriever, which operates on a propositional language, handles only yes/no queries, the second also handles wh-queries, and the third allows quantifiers in the KB and the query. Each is shown to be, in some sense, the strongest retriever that meets the no-chaining restriction.

The third retriever forms an excellent basis for integrating a specialized set of inferences that chain in a controllable manner. This is achieved by incorporating taxonomic inference, such as inheritance, to form the fourth retriever, an idealized version of the retriever incorporated in the ARGOT natural language dialogue system. It is characterized by its ability to infer all consequences of its taxonomic knowledge without otherwise chaining.

Table of Contents

List of Figures

Chapter 1

Introduction

One truism of artificial intelligence is that an intelligent system must have vast amounts of knowledge of its domain. Some of this knowledge appears to be maintained independently of its use. For example, I know that certain English words are considered rude. I can use this knowledge in achieving my goals, thereby behaving differently when I choose to insult from when I choose to be polite. I can also use this knowledge to recognize the intentions of other speakers. Furthermore, I can verbalize this knowledge in order to inform a non-native speaker of our conventions. Knowledge of this nature is often called "declarative," though the word is somewhat ill-defined.

Accordingly, artificial intelligence reasoning systems commonly contain a large corpus of declarative knowledge, called a knowledge base (KB), and provide facilities with which the system's components can retrieve this knowledge. An example of such a system, the ARGOT dialog participation system, is outlined in the next section.

This thesis addresses the problem of building a knowledge retriever and obtaining a thorough understanding of it. Though my study of retrieval began by designing and implementing the retriever in ARGOT, and the retriever examined in this thesis forms the core of the ARGOT retriever, the goal of the work is not primarily the construction of a particular retriever. Rather, its goal is more generally to gain an understanding of the very nature of retrieval and to produce techniques useful in addressing this and, hopefully, related problems.

I intend to attain these general goals while constructing a specific retriever by carrying out the construction in a principled way. First, I elucidate certain intuitions about the nature of retrieval and put forth criteria that a retriever must satisfy. Then I formally specify a retriever that captures the intuitions and meets the criteria. Finally, I design an algorithm that implements the specification.

In sum, the principal contribution of this thesis is the transformation of retrieval from an ill-defined unstudied process to a formally-defined and well-studied one.

1.1. An Overview of ARGOT's Organization

A very brief overview of ARGOT's organization suffices to illustrate the structure of a system containing a KB and knowledge retriever and to show how these components are used. Allen, Frisch and Litman (1982) describe ARGOT in more detail.

ARGOT is designed to play the role of a computer operator partaking in extended English dialogs with computer users. As depicted in Figure 1.1, the system is divided into three concurrently-running modules: a task goal reasoner, a communicative goal reasoner, and a linguistic reasoner. All declarative knowledge used by the three modules is stored in a common knowledge base and is accessed only through a knowledge retriever. Hence, the only view of the KB available to the system's modules is that provided through the query-processing facilities of the retriever. The internal structure of the KB is completely hidden.

The reasoning modules of ARGOT are not general-purpose reasoning systems but rather are highly specialized for the tasks they perform. On the other hand, the retriever is not specialized for any particular task such as linguistic reasoning, though it is able to do some taxonomic and temporal reasoning.

1.2. Knowledge Retrieval as Inference

At all points throughout this thesis I keep in mind that a KB is not only a data structure, but also a representation. By this I mean that the KB makes a set of assertions, sentences to which a semantics attributes truth conditions.

In response to a query a retriever returns one or more sentences. (A sentence that can be retrieved from a KB by *some* query is said to be *retrievable.)* In doing this, the retriever must respect the semantics of the language it operates on. More precisely, a retriever should return a sentence only if the truth of the sentence is assured by the truth of the KB, that is, only if the sentence is entailed by the KB.

These considerations lead to the viewpoint that retrieval is inference, which is taken to be a mechanism that derives logical consequences. But what kind of inference is retrieval? How much and what kind of inference capability should be packaged up in a retriever?

Let us address these questions by first considering the extreme position of making a retriever as strong as possible. Such a retriever would be able to retrieve all logical consequences of the KB. An example of this approach is the KRYPTON knowledge representation system (Brachman, Fikes and Levesque, 1983; Brachman, Gilbert and Levesque, 1985), which employs a complete inference mechanism known as theory resolution (Stickel, 1985). However, in discussing their design of KRYPTON, Brachman, Gilbert and Levesque (1985) express dissatisfaction with this choice of inference mechanism:

> We would no doubt have used a more computationally tractable inference framework than full first-order logic if an appropriate one were available... the full first-order resolution mechanism is, in a sense, too powerful for our needs.

Indeed, in any language as expressive as the first-order predicate calculus determining whether one sentence entails another is only semi-decidable. If we design a system that relinquishes control to its retriever then we must demand that the retriever returns control and that it quickly does so.

To design a retriever that computes all, or even most, of the logical consequences of a KB is to put the muscle of the system in the wrong place. The power of the system belongs in the special-purpose modules of the system, which can be built with the necessary domain-dependent control structures, not in the knowledge retriever, which is a domain-independent inference engine.

Efficiency rather than power is the primary consideration in designing a retriever. A system can compensate for a retriever that is efficient but weak by performing computations itself, but it cannot compensate for a retriever whose excessive power leads to inefficiency.

The above considerations lead to the conclusion that retrieval is limited inference. This raises the question to which we now turn: what inferences should a retriever perform?

1.3. Retrieval as Pattern Matching

A common intuition, whose prevalence can be witnessed by examining the range of retrievers in use, is that retrieval is fundamentally a pattern-matching operation. According to this intuition a KB consists of a set of data objects and a query supplies a target pattern to which the retriever responds by reporting which data objects match the pattern. Whether or not a particular data object matches the target is independent of the other data objects. Hence, in a parallel implementation each data object can be realized by a distinct processor and there is no need for communication between them. The idea of using pattern matching to access a memory structure is a familiar one, present, for example, in the notion of a content-addressable memory.

A strict pattern-matching system, such as a content-addressable memory, never combines two or more data objects in order to match a target successfully. Doing so would require communication between the processors in a content-addressable memory. I call such an operation "chaining."

In general, chaining is the operation of combining two or more pieces of a representation together in order to derive a third. The archetypal form of chaining occurs in applying the rule of modus ponens to infer Q from P and $P \rightarrow Q$.

The simplest form of pattern matching occurs in matching an object against itself. Hence, if retrieval is pattern matching, a retriever should be able to retrieve everything that has been added explicitly to the KB. A retriever that can do this is said to satisfy the ***verbatim retrieval criterion***. For example, if I add "$P \vee Q$" to a KB, then its retriever should report "yes" when I query "$P \vee Q$". It should do this readily by matching "$P \vee Q$" to itself.

Contrast this, the simplest case of pattern matching, with the operation of a resolution theorem prover. Using a refutation strategy, it replaces the problem of proving that $P \vee Q$ entails $P \vee Q$ with the problem of proving that $\{P \vee Q, \neg P, \neg Q\}$ is unsatisfiable. It can solve this problem by finding one of the resolution refutations of Figure 1.2. Each of these refutations contains two applications of the resolution inference rule. Every application of this inference rule resolves two clauses together to derive a third and hence involves chaining.

To increase their power, pattern matchers are often extended with rewrite or deduction rules. One such system is SNePS (Shapiro, 1979), a semantic-network system that a user accesses by specifying a target network-pattern to be matched against the network in the KB. SNePS also allows the user to specify backward chaining rules that are used to attempt to rewrite target patterns. [1] Database systems also are frequently extended with facilities to handle rewrite rules; such systems are called deductively-augmented data bases. A final example of the incorporation of rewrite rules into a pattern matcher is the ubiquitous use of inheritance in semantic-network systems. Unlike the rewrite rules used by SNePS or deductively-augmented data bases, which are added to the system by the user, the rewrite rules that constitute inheritance are built directly into semantic-network retrievers.

We are now in a position to answer the question, "What is retrieval?" *Retrieval is an inference process, limited in such a way that it is fundamentally a pattern-matching process, though it may be extended to do a bounded amount of some specialized form of chaining.* Though I take this as being an accurate characterization of retrieval, it is an intuitive characterization, not a rigorous or precise one. This is appropriate because the notion of retrieval is intuitive rather than technical. The intuitive nature of this characterization arises from its formulation in terms of the notion of pattern matching, itself an intuitive notion.

Let us examine our intuitions about pattern matching more closely. The weakest pattern-matching retriever can retrieve only what is contained explicitly in the KB. However, pattern matchers can often succeed at

- retrieving $P \vee Q$ from a KB containing $Q \vee P$, and
- retrieving Q from a KB containing $P \wedge Q$.

While our intuitions clearly allow us to call these retrievals pattern matching, our intuitions get murky when we try to see how far the notion extends. Which of these actions should be called pattern matching:

- retrieving $P \vee Q$ from a KB containing Q,
- retrieving $\neg\neg P$ from a KB containing P, and
- retrieving $P \rightarrow Q$ from a KB containing $\neg P \vee Q$.

Though it is not clear where the intuitive notion of pattern matching ends, it is clear that retrievals involving chaining, such as

- retrieving Q from a KB containing P and $P \rightarrow Q$,

are *not* pattern matching.

Unlike pattern matching, chaining becomes a precise term once a method of breaking a representation into quanta has been determined. Whether or not a given retriever performs chaining is then a cut and dried question. Because the notion of chaining is precise and is closely aligned with the intuitive notion of pattern matching, I henceforth use no-chaining,

[1] SNePS also allows forward chaining rules to be specified.

rather than pattern matching, as the defining characteristic of retrieval.

The effect of this *no-chaining restriction* depends critically on the granularity at which knowledge is quantized. At one extreme, if the entire KB is considered to be a single quantum, the no-chaining restriction becomes vacuous; there can't be any chaining simply because there aren't two quanta to be chained together. At the other extreme, if each quantum is merely an atomic sentence then only verbatim retrievals can be performed; the only atomic sentence entailed by an atomic sentence is the sentence itself.

If the elimination of chaining is to eliminate all inference that does not correspond intuitively to pattern matching, then the KB must be divided into fine-grained quanta. As an illustration, if the no-chaining restriction is to disallow

- retrieving Q from a KB containing $P \wedge (P \rightarrow Q)$,

then the KB should be divided into two quanta, P and $P \rightarrow Q$. I shall return to the issue of quantization in Chapter 2, where the design of the first retriever is undertaken.

1.4. Thesis Synopsis

The principal technical result of this thesis is the formal specification of a knowledge retriever that performs *all* inferences that don't require chaining, *all* inferences that involve taxonomic information, but *no* others. The specified retriever is a highly specialized inference engine; it does all chaining of a certain sort but no other chaining. Relative to the way that knowledge is quantized, this is the *strongest* retriever that does no chaining other than with taxonomic information.

This retriever operates on a first-order logic. While some may argue that this language is not sufficiently expressive for certain AI tasks such as natural language processing, it is expressive enough to make the retrieval problem difficult. Most notably, entailment is undecidable in this logic.

This retriever is specified as the last in a series of four retrievers, each of which extends the capabilities of the previous one. The four specifications are presented successively in Chapters 2 through 5 of this document. Accompanying each specification is a rigorous investigation into its properties, a presentation of an efficient, terminating algorithm, and a proof that the algorithm meets the specification.

The first retriever, which operates on a propositional language, handles only yes/no queries. The second retriever additionally handles wh-queries; its algorithm introduces unification into the simple algorithm of the first retriever. An important result shows that even when there are an infinite number of answers to a wh-query, the set of answers can be finitely characterized.

The third retriever extends the second by allowing quantifiers in the KB and the query. First, it is shown that the obvious, straightforward treatment of quantifiers results in a retriever that is not guaranteed to terminate. Closer examination of the difficulty reveals that this treatment of quantifiers subtly violates the no-chaining restriction. A new specification is presented, which uniformly respects the no-chaining restriction throughout its treatment of the entire language, including the quantifiers. Analysis of the relationship between quantifiers and wh-queries reveals how the second retrieval algorithm can be used to implement this third specification.

Each of these first three retrievers meets the no-chaining restriction; moreover, for the class of problems handled each is the *strongest* retriever that meets the restriction, modulo the way that knowledge is quantized. If the thesis were to end here, it could aptly be entitled "The Logic of No-Chaining."

The fourth retriever integrates taxonomic inference such as inheritance into the pattern-matching inference of the third retriever. The algorithm for this fourth retriever is interesting in that it uses the taxonomic information *solely* during unification. A theorem is presented that states the conditions that are necessary and sufficient to justify this computational technique. The technique and the theorem justifying it are both general enough to be used to add a taxonomic component to almost any computational system that works with schematic variables, including rewrite systems, grammars and their parsers, theorem-provers, and logic-programming systems. This fourth retriever forms the core of the retriever incorporated in ARGOT.

1.5. A Method for Specifying Retrievers

Rather than specify how a retriever operates, this thesis concentrates on specifying what a retriever computes. This is done by specifying a retrievability relation that determines what sentences are retrievable from what KB's, just as a provability relation determines what sentences are provable from what sets of axioms. Thus, retrievability is a relation between knowledge bases and sentences. If knowledge base kb and sentence q stand in this relation then we say that q is retrievable from kb and write $kb \rightarrow_R q$. Thus, the problem of specifying a retriever comes down to one of specifying a retrievability relation. This section puts forward a model-theoretic specification method that is used throughout this work. [2]

Each of the four retrievers of this thesis operates on a language whose semantics is given by a standard Tarskian (1935) model theory. Each model theory yields an entailment relation which, analogous to a provability or retrievability relation, defines what sets of sentences entail what sentences. Entailment relations are well-suited for use in specifications because they are precise, often have simple definitions, and abstract away from all issues of formal syntactic

[2] Frisch (1985a) provides a more-general discussion of this technique.

operations. However, Tarskian entailment cannot be used to specify retrievability; what is retrievable from a KB is but a small subset of what is entailed by a KB.

The approach advocated here for specifying retrievability is to produce another model theory whose entailment relation is weakened[3] in such a way that the retriever is a sound and complete inference engine with respect to it. Many people initially find this approach quite odd. They are accustomed to thinking of a model theory as specifying what can be concluded validly from what—in some sense, as a competence theory of inference. I suggest that those who are comfortable with this viewpoint consider the weaker model theory as a performance theory of inference. Other people are accustomed to thinking of a model theory as a way of assigning meaning to symbols and are skeptical of producing a new meaning assignment. But I am not suggesting that the original model theory be discarded; on the contrary, it is still a valuable device in the study of meaning. The new model theory can be thought of as providing an additional meaning assignment. If the retriever is working under this alternative meaning then it is a complete inference engine. Hence, the symbols mean one thing to us and another to the retriever. According to our theory of meaning the retriever is incomplete but according to its weaker theory of meaning it is complete.

How can these new, weaker model theories be produced and what is their relationship to the unweakened model theory? To answer the question consider a model theory as laying down a set of constraints on what constitutes a model. Of all (mathematical) objects, only those that satisfy the constraints qualify as models. A model theory also associates with each model a valuation, a total function from sentences to their truth values. Hence, a model theory constrains the range of valuations that can be generated. In a standard propositional logic, for example, these constraints ensure that any valuation that assigns two sentences True, also assigns their conjunction True. The entailment relation associated with a model theory is a product solely of the range of valuations that the model theory generates. Relaxing the constraints produces a new model theory, one that may generate additional valuations. *No matter how the constraints are relaxed, the new model theory must have a weaker entailment relation than the original.* That is, if $\models_1$ and $\models_2$ are entailment relations and $\models_2$ is obtained by relaxing the model theory for $\models_1$, then $\alpha \models_2 \beta$ implies $\alpha \models_1 \beta$. To see this, observe that a valuation can serve only as a counterexample to a claim that one sentence entails another; hence if none of the valuations from the relaxed model theory are counterexamples then certainly none from the original model theory are.

[3] One entailment relation is *weaker* than another if the inferences sanctioned by the first are a subset of those sanctioned by the second

1.5.1. A Toy Example

Consider a program that reasons about an arbitrary equivalence relation named "r". A user communicates with the program, making assertions and queries, each of which specifies a sentence of the form $r(\alpha,\beta)$, where α and β are symbols drawn from some lexicon.

This program can be specified in terms of the symbolic manipulations it performs. There are many conceivable specifications but for the sake of argument let us say that the program works by maintaining a collection of disjoint sets. Initially there is a unit set for each symbol in the lexicon. Whenever the user asserts $r(\alpha,\beta)$ the program combines the sets that contain α and β into a single set. To respond to the query $r(\alpha,\beta)$ the program simply determines whether the elements α and β are in the same set. There are many well-studied algorithms for performing this set-union task (Tarjan and van Leeuwen, 1984), the best of which can process a series of n assertions and queries in slightly greater than $O(n)$ time.

To the user of the system this specification is too detailed and too concrete. He does not need to know, nor does he care, whether the program works one way or another. At the level of abstraction with which the user is concerned the various implementations are all identical. (Of course, for other concerns, such as implementation, there is a world of difference between different specifications.) A more abstract, non-procedural definition of this reasoning system can be obtained by replacing the question "What does the reasoner do?" with "Given a set of previously-asserted sentences, what query sentences succeed?" At this higher level of abstraction the various set-union algorithms are all equivalent.

In response to the question of what queries succeed, it can be shown that the program described above answers "yes" to a query if, and only if, it legitimately can do so based on what it has been told. That is, it answers "yes" if, and only if, the queried sentence is entailed by the set of asserted sentences. Forgive my pedantry while I spell out the obvious details of the model theory that gives rise to the entailment relation for this language; these details will be valuable in considering how to relax a model-theoretic specification.

Each of the model theories discussed in this thesis is given a name. The one presented next is called "E". In cases where confusion could arise, terms like "E-model" and "E-entailment" and symbols like $\models_E$ are used to indicate which model theory is under consideration.

An E-model is a pair $\langle D,A\rangle$ where D is a non-empty set of individuals called the domain and A is a function that maps every symbol in the lexicon to an element of D and maps r to a binary relation over D such that:

(1) $A(r)$ is reflexive,

(2) $A(r)$ is symmetric, and

(3) $A(r)$ is transitive.

The valuation associated with $\langle D,A\rangle$ is the function that takes each sentence of the form $r(\alpha,\beta)$ to True if the relation $A(r)$ holds between $A(\alpha)$ and $A(\beta)$, and to False otherwise.

These valuations can be used in the usual fashion to define the notions of E-satisfiability, E-validity, and E-entailment for this language.

This model theory serves two purposes in analyzing the program. First, it provides a rigorous semantics for the language that the program manipulates and in doing so defines entailment for the language. Second, it is used in specifying what the program computes by stipulating that it responds "yes" if, and only if, the queried sentence is E-entailed by the asserted sentences. In the case of this program the two uses go hand-in-hand because the program is a sound and complete inference engine. But it is important to distinguish between these two uses of a model theory as we turn our attention to a reasoning program that is not complete.

Suppose that for some reason we were not happy with a program that required slightly greater than $O(n)$ time to process a series of n assertions and queries. (I told you this was a toy example!) Furthermore, suppose that we were willing to replace the set-union algorithm with the following algorithm, which is much weaker but slightly faster. Whenever $r(\alpha,\beta)$ is asserted, the program adds the pair $\langle\alpha,\beta\rangle$ to an associative store. The program responds "yes" to the query $r(\alpha,\beta)$ if, and only if, alpha and beta are identical or the associative store contains either $\langle\alpha.\beta\rangle$ or $\langle\beta,\alpha\rangle$.

This program is incomplete with respect to E. For example, if only $r(a,b)$ and $r(b,c)$ have been asserted, the query $r(a,c)$ will not result in "yes" even though the queried sentence is E-entailed by the two asserted sentences. E *still* gives a semantics for the language manipulated by the program but it no longer specifies what the program computes. However, there is a weaker model theory—call it E^w—whose entailment relation does specify the input/output relation of this program. E^w is identical to E except that constraint (3), which says that $A(r)$ must be transitive, is eliminated. With respect to E^w the program is a sound and complete inference engine—though admittedly soundness and completeness are normally taken to be with respect to a model theory that specifies the meaning of the language.

Every model in E is also a model in E^w, but not vice-versa. For example, consider the model $\langle D',A'\rangle$ where $D'=\{1,2,3\}$ and A' is such that:

$A'(a) = 1$

$A'(b) = 2$

$A'(c) = 3$

$A'(r) = \{\langle 1,1\rangle,\langle 2,2\rangle,\langle 3,3\rangle,\langle 1,2\rangle,\langle 2,1\rangle,\langle 2,3\rangle,\langle 3,2\rangle\}$

This model respects constraints (1) and (2) but not constraint (3). The valuation generated by $\langle D',A'\rangle$ is not generated by any E-model, hence $\models_{E^w}$ is strictly weaker than $\models_E$. Returning to the example previously cited, $\langle D',A'\rangle$ demonstrates that $r(a,b)$ and $r(b,c)$ do not E^w-entail $r(a,c)$.

Though this example program specification is quite simple it has illustrated many of the major points about the method of specifying programs with model theory.

1.6. Requirements on Retrievability

There are three restrictions that I place on any retrievability relation. Each of these already has been discussed more or less explicitly, but here they are made explicit and precise by presenting them in terms of the restrictions they impose on a retrievability relation.

If *kb* is a set of sentences and *q* a single sentence of some language whose Tarskian entailment relation is $\models_T$ then any retrievability relation $\rightarrow_R$ for that language must satisfy:

soundness: $kb \rightarrow_R q$ only if $kb \models_T q$.
verbatim retrieval: if $q \in kb$, then $kb \rightarrow_R q$.
decidability: $\rightarrow_R$ is a decidable relation.

The first requirement demands that no sentence can be retrieved from a KB unless it is entailed by the KB, while the second demands that sentences explicitly in the KB are retrievable. Both of these requirements are met by any retrievability relation that is the entailment relation of a model theory obtained by relaxing T. As previously argued, relaxing constraints in a model theory can only weaken its entailment relation and therefore the soundness requirement is met. The verbatim retrieval requirement quite clearly is met by *any* entailment relation.

It is a virtue of the model theoretic specification technique that it imposes constraints on the relations that can be specified, some of which are demanded independently by the nature of the relations that we are attempting to specify. Consequently, relations that fail to satisfy the soundness and verbatim retrieval requirements cannot even arise in this study. This contrasts with earlier work (Frisch and Allen; 1982) employing a more syntactic form of specification in which it was necessary to prove that each specified retriever met these requirements.

Given that the retrievability relations of this thesis are produced by the advocated model-theoretic method, only the decidability requirement need concern us. This requirement, which ensures that the retriever can be realized by an effective procedure that is guaranteed to terminate, is weaker than it ideally should be—that the retriever could be realized by a procedure requiring only some small amount of computational resources. As previously discussed, the first three retrievers attempt to achieve this required efficiency through the elimination of chaining. Achieving decidability in the fourth retriever requires that all necessary taxonomic reasoning can be performed with only a bounded amount of chaining.

The three requirements discussed in this section can be captured poetically, as well as formally:

Insist what's explicit is found
Within computational bound.
And what's retrievable
Must be believable,
So make sure that inference is sound.

1.7. Other Methods for Limiting Inference

The computing literature contains a number of approaches to limiting inference. This section reviews several of these and argues that none of them is appropriate for limiting inference in a retriever. This should not be surprising, for each of the approaches was developed for purposes other than retrieval.

One way of limiting inference is to restrict the expressiveness of the representation language used to express the knowledge in the KB. Such restrictions can simplify the decision of whether a given fact is entailed by the facts in the KB. This is the approach taken to yield efficient retrieval in data bases. For example, even in the area of logic data bases where the emphasis is on more expressive languages, data bases often are limited to the Horn-clause subset of First-Order Predicate Calculus.[4] This restriction precludes expressing "Either John or his brother has the money" without expressing which one has the money. This approach to simplifying retrieval can also take the form of making assumptions about the domain and its relationship to the representation language. Common assumptions are that the domain is finite, that each individual symbol in the language denotes a distinct individual (E-saturation or unique name assumption; Reiter (1980; 1984)), that each individual is denoted by some individual symbol (Domain closure; Reiter (1980; 1984)), and that everything that cannot be proved true is false (Closed World Assumption; Reiter (1978; 1984)). The general acceptance of this approach to achieving efficient retrieval distinguishes the field of data bases from the field of AI knowledge representation.

Assumptions of the sort described above should not be built into an AI knowledge representation language. To do so would restrict the set of situations in which a system could perform intelligently. These assumptions are just not true of our everyday world and its relationship to any system possessing common sense.

A representation should define a set of valid inferences that *could* be made, not those that *are* made. Even if the retriever only makes a small portion of all valid inferences the remaining possibilities must be available for the reasoner to consider.

[4] Gallaire, Minker, and Nicolas (1978) overview the field of logic data bases in general, and define Horn clauses in particular. Kowalski (1979) treats Horn clauses in depth.

Another common approach to limiting inference is to restrict the amount of resources used in the computation (Norman and Bobrow, 1975; Bobrow and Winograd, 1977; Robinson and Sibert, 1981). This can be done by restricting computation time, the total number of inference steps taken, or the depth of the inference. These approaches are unsuitable for knowledge retrieval because they limit all forms of inference uniformly. For example, if inference is limited to a depth of 5, then properties cannot be inherited down 6 levels of a type hierarchy. In general, there may be some kinds of inference that we want to be computed completely and others that we want to be ignored completely. A methodology for limiting inference for retrieval should provide the designer with enough control to pick and choose the inferences that he wants to be performed.

A further class of limited inference systems consists of the incomplete theorem provers that are fairly common in the literature: for example, Brown's (1978) system. Typically, these systems are not guaranteed to terminate, and often fail to meet the verbatim retrieval criterion.

A method of limiting inference that has been suggested to me is to find a traditional proof system in the mathematical literature consisting of a set of axioms and inference rules and then eliminate some of these inference rules and/or axioms. But there is no reason to believe that the distinctions between the kinds of inference drawn by these inference rules and axioms are the distinctions that I want to draw.

1.8. A Note on Notation

I have attempted to use standard notation and terminology as far as possible. Most of the notation and terminology is defined when it is first used. However, it is worth introducing some at the very outset:

- Recalling that $\alpha \models \beta$ means that no model both satisfies α and falsifies β, a model that satisfies α and falsifies β is called a *countermodel* to $\alpha \models \beta$.
- A *literal* is an atomic formula or its negation. The former are called *positive* literals and the latter, *negative* literals. If α is an expression or a set of expressions, $LITERALS(\alpha)$ is the set of all literals occurring in α.
- If ϕ is a formula then its *complement,* written ϕ^c, is defined by:

$$\begin{aligned} \phi^c &= \psi \text{ if } \phi = \neg\psi \\ &= \neg\phi \text{ otherwise} \end{aligned}$$

- A model theory is said to be decidable or undecidable according to the decidability of its entailment relation.
- As previously stated, every model theory discussed in this thesis is given a name. T is the Tarskian model theory. It names the Tarskian model theory for whatever language happens to be under discussion.

- The least upper bound of a set X is written as $\bigsqcup X$.
- In introducing a formula ϕ, I may write $\phi[x_1, \ldots, x_n]$. Subsequently, I write $\phi[t_1, \ldots, t_n]$ to refer to that formula which results from replacing all occurrences of x_i in ϕ by t_i (for $1 \leq i \leq n$).
- If ϕ is a formula containing free variables $x_1, \ldots, x_n$, then its *universal closure,* denoted by $\forall\phi$, is $\forall x_1, \ldots, x_n\ \phi$, and its *existential closure*, denoted by $\exists\phi$, is $\exists x_1, \ldots, x_n\ \phi$.

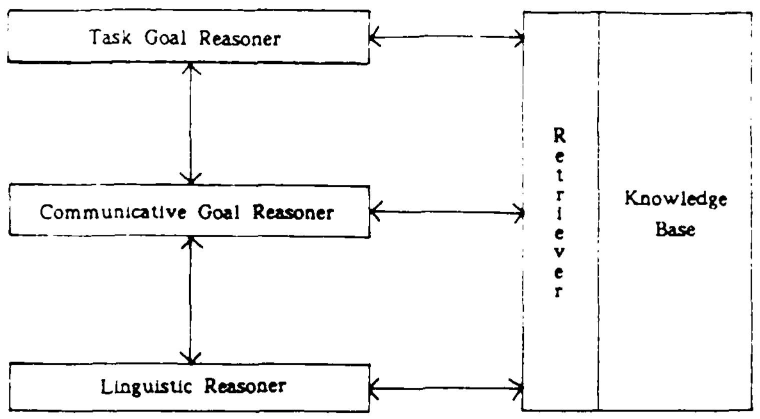


Figure 1.1: The Organization of ARGOT

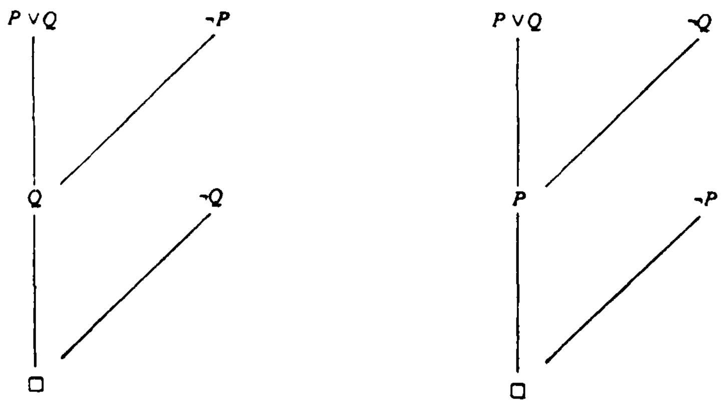


Figure 1.2: Resolution Refutations of {P∨Q, ¬P, ¬Q}

Chapter 2

A Retriever for a Propositional Language

This chapter develops and studies a retriever for a propositional language whose truth-functional semantics is given by the standard Tarskian (1935) model theory. The retrievability relation developed is the entailment relation of RP, a model theory obtained by relaxing the Tarskian model theory.

Even within the confines of a propositional language one encounters the most fundamental issues in the study of retrieval: How can a representation be quantized? How can a Tarskian model theory be relaxed to obtain a model theory that does not allow these quanta to be chained? After chaining is eliminated, what sort of inferences remain? What properties does a "logic of no-chaining" have and how do they compare to the properties of the Tarskian logic from which it was derived?

The principal results of this chapter include:

- a workable definition of a "quantum" of representation,
- the specification of RP, a propositional logic that is a relaxation of T,
- a proof that no form of chaining is valid in RP,
- a proof that, modulo the way that the representation is quantized, RP is the strongest logic that allows no chaining, and
- a specification of a retrieval algorithm that meets this model-theoretic specification, i.e., a decision procedure for RP.

2.1. The Quantifier-Free Predicate Calculus

The retriever studied in this section operates on the sentences of a language called "Quantifier-Free Predicate Calculus" (QFPC). That is, throughout this chapter a KB consists of a finite set of QFPC sentences and a query specifies a QFPC sentence to be retrieved. Therefore, the retrieval problems considered in this chapter are those that can be represented as a finite QFPC sequent, a sequent whose antecedent is a finite set of QFPC sentences and whose consequent is a single QFPC sentence.

Syntactically and semantically, QFPC is identical to First-Order Predicate Calculus except that QFPC has no quantifiers. Briefly, the lexicon of QFPC consists of a set of *variables,* a set of *function symbols,* and a set of *predicate symbols.* In the usual manner, variables and function symbols can be combined to form *terms,* which can be combined with predicate symbols to form *atomic formulas,* which in turn can be combined with the logical connectives to form *molecular formulas.* As usual, a *sentence* is a formula with no free variables. Therefore, since QFPC has no quantifiers, a QFPC sentence has no variables whatsoever.

It may appear curious that QFPC allows variables in its formulas even though the retriever only deals with variable-free sentences. However, there is reason to this madness. Accommodating variables into the analysis of this chapter adds little complexity but greatly simplifies matters in Chapter 4 when the retriever is extended to handle quantifiers. For example, this chapter presents a theorem concerning the application of a normal-form transformation to formulas. Because this result concerns all formulas—not just sentences—it is applicable to the more-general circumstances that arise in Chapter 4.

This is a good place to reiterate that I never speak of the value of an open formula relative to a model; it only has a value relative to a model *and* an assignment of values to variables (henceforth simply called a value assignment). A sentence, however, does have a value relative to a model, and can be satisfied or falsified by a model. Accordingly, only sentences participate in entailment relations—including $\models_{RF}$, which specifies the retriever of this chapter.

2.2. A Matter of Fact

Recall my original suggestion that a retriever should operate by dividing the KB and query into quanta called "facts" and then performing retrieval on a fact-by-fact basis. In other words, a query succeeds if, and only if, each of its facts is retrievable from a single fact in the KB. However this in itself does not necessarily lead to an efficient retriever since it may still be difficult to decide whether a single fact is retrievable from a single fact. Two extreme approaches could be used to obtain an easily-decidable retrievability relation between facts. One is to allow facts to be complex objects and to make retrievability between facts be a weak relation in comparison to T-entailment. The other approach is to allow facts to be only simple objects and to make retrievability between facts be a strong relation in comparison to T-entailment. If the first route is taken, a version of the original problem remains: How should T-entailment be weakened to yield an efficient retrievability relation over the set of facts?

I pursue the second route. Facts are defined to be so syntactically simple that retrievability over the set of facts can be taken as T-entailment. A retriever designed in this way occupies a privileged position; modulo the definition of fact, it will be the *strongest* retriever that does no chaining.

These considerations impose three criteria on the way that facts and the RP model theory are defined. Let us now make these criteria explicit. If ϕ_1 and ϕ_2 are facts and Φ is a set of facts then:

No chaining: $\Phi \models_{RP} \phi_1$ iff for some $\phi \in \Phi$ $\phi \models_{RP} \phi_1$.

Strength: $\phi_1 \models_{RP} \phi_2$ iff $\phi_1 \models_T \phi_2$.

Efficiency: There is a simple algorithm that decides whether $\phi_1 \models_T \phi_2$.

Taken together, the Strength Criterion and the Efficiency Criterion demand that T entailment between facts must be easily decidable. Since deciding T entailment between arbitrary sentences of QFPC is intractable, facts must be expressively weaker than sentences in general; that is, it must be that there are sentences that are not T-equivalent to any fact. The two most obvious choices are to take a fact as a disjunction of variable-free literals or as a conjunction of variable-free literals. I have chosen the former. I know of no reason why the other choice is not viable and, though not pursued here, an investigation of the consequences of that choice would be worthwhile. However, of the two choices, a disjunction of literals provides a more natural basis for knowledge representation.

The following theorem assures us that with the above definition of "fact" it is possible to satisfy the Strength Criterion without violating the Efficiency Criterion.

T-Decision Theorem for Facts
For any facts, ϕ_1 and ϕ_2, $\phi_1 \models_T \phi_2$ iff either
(1) ϕ_2 has complementary literals, or
(2) $LITERALS(\phi_1) \subseteq LITERALS(\phi_2)$.

Proof
If clause: Assume Condition (2). From the definition of $\vee$ observe that a T-model satisfies a disjunction iff it satisfies one of the disjuncts. Therefore, any T-model satisfying ϕ_1 also satisfies ϕ_2; i.e., $\phi_1 \models_T \phi_2$. On the other hand, assume Condition (1). From the T-truth table for $\neg$ observe that each literal or its complement is satisfied by any T-model. Thus, (by the argument above) any disjunction containing complementary literals is satisfied by every model, and therefore $\phi_1 \models_T \phi_2$.
Only-if clause: Assuming neither Condition (1) nor (2) is satisfied, I construct M, a T-model that satisfies ϕ_1 but falsifies ϕ_2. Let M assign False to every literal in ϕ_2. This is possible since by assumption no literal occurs both positively and negatively in ϕ_2. Therefore M falsifies ϕ_2. An obvious consequence of the initial assumption is that ϕ_1 contains some literal not contained in ϕ_2. Call it L. If L^c is one of the disjuncts in ϕ_2 then M assigns False to L^c and hence True to L. Otherwise, we are free to let M assign True to L. In either case, M assigns True to L and therefore, by the definition of $\vee$, satisfies ϕ_1. ■

The two syntactic conditions for T-entailment can be easily computed. Even if facts are encoded as unordered lists, this decision can be made in O(n log n) time, where n is the sum of the lengths of the facts. This could be reduced further by a better encoding of facts. Proof complexity is another important measure. A proof that one fact T-entails another would have to demonstrate either that every member of a certain set of literals was contained in another or that a certain set contained complementary literals. The size of each of the necessary demonstrations is proportional to the size of the facts involved.

2.3. Defining RP

With the definition of "fact" in hand we are ready to relax the constraints on T in order to produce RP. In doing so, we can safely ignore the Efficiency Criterion because of the previous theorem, and thereby concentrate on satisfying the No-Chaining and the Strength Criteria. However, this is tricky because these criteria apply opposing forces; the No-Chaining Criterion demands a logic that is weak in a certain way, while the Strength Criterion demands a logic that is strong in another way.

It is tempting to try to produce the specification of the relaxed model theory by following the tactic used in Section 1.5.1 of simple textual deletion of some constraint on what constitutes an unrelaxed model. However, in the case of T it is not so straightforward. E specifies three constraints on the relations that can be assigned to the symbol r and thus prevents the atomic sentences of the language from obtaining certain combinations of truth values. (Recall that all sentences in the object language of E are atomic.) Relaxing E to obtain E^w involves deleting one of the three constraints, allowing the atomic sentences to be given additional combinations of the two truth values.

Unlike E, T places no constraints on what a model can assign to a predicate symbol and therefore the atomic formulas of the language can be assigned *any* combination of the two truth values. So the strategy of generating additional valuations by giving the atomic sentences more combinations of the two truth values cannot be pursued in this case. This leaves two options: either allow atomic sentences to be mapped to values other than True or False or modify the way values are assigned to molecular formulas. This thesis pursues the first strategy. Elsewhere (Frisch, 1985a) I have specified a nearly identical retriever using the second strategy.

Examination of why P and $\neg P \vee Q$ T-entails Q provides the insight used to derive the new model theory, RP. Consider this three-step argument that $P, \neg P \vee Q \models_T Q$:

(1) Assume that P and $\neg P \vee Q$ are both satisfied by a certain model.

(2) Since P is satisfied, $\neg P$ isn't.

(3) Consequently, if the model is to satisfy $\neg P \vee Q$, as assumed, it must satisfy Q.

As far as chaining is concerned, step (2) is the crucial one; it connects P and $\neg P \vee Q$. The validity of the step rests on the assumption that a model satisfies only one of P and $\neg P$—a justified assumption in T where a model assigns each sentence either True or False, but never both. RP relaxes the restriction that the assignments of True and False are exclusive by allowing each sentence to be assigned a *non-empty* subset of {True, False}. Hence, RP has three truth values: {True}, {False} and {True, False}. We will see that this modification admits models that satisfy both P and $\neg P$, thus eliminating modus ponens as a sound rule of inference.

Let us make this precise. Like a T-model, an RP-model is a pair $\langle D,A \rangle$ where D—the domain—is a non-empty set and A is an assignment of appropriate semantic objects to the non-logical symbols of QFPC. As in a Tarskian model, A assigns to every n-ary function symbol a function from D^n to D. However, unlike a Tarskian Model, A assigns to every n-ary predicate symbol a function from D^n to {{True}, {False}, {True, False}}. In comparing RP to T the difference between True and {True} and between False and {False} always will be ignored.

The exclusivity of True and False is built implicitly into the usual semantic equations that determine how T recursively assigns values to molecular formulas. Consider, for example, (2.1) and (2.2), the semantic equations for disjunction and negation. Here, $[\![\phi]\!]^{M,e}$ is the truth value of formula ϕ relative to model M and value assignment e.[1]

$$\begin{aligned}[\![\alpha \vee \beta]\!]^{M,e} &= \text{True if } [\![\alpha]\!]^{M,e} = \text{True or } [\![\beta]\!]^{M,e} = \text{True} \\ &= \text{False otherwise}\end{aligned} \tag{2.1}$$

$$\begin{aligned}[\![\neg\alpha]\!]^{M,e} &= \text{True if } [\![\alpha]\!]^{M,e} = \text{False} \\ &= \text{False otherwise}\end{aligned} \tag{2.2}$$

Notice that in these equations the assignment of False is based on the non-assignment of True. Let us now assume that formulas can be assigned a *set* of values—{True} and {False} in the Tarskian case—and define the assignment of True and False independently of each other. (2.1) and (2.2) can be written equivalently as (2.3) and (2.4).

$$\begin{aligned}&\text{True} \in [\![\alpha \vee \beta]\!]^{M,e} \text{ iff True} \in [\![\alpha]\!]^{M,e} \text{ or True} \in [\![\beta]\!]^{M,e} \\ &\text{False} \in [\![\alpha \vee \beta]\!]^{M,e} \text{ iff False} \in [\![\alpha]\!]^{M,e} \text{ and False} \in [\![\beta]\!]^{M,e}\end{aligned} \tag{2.3}$$

$$\begin{aligned}&\text{True} \in [\![\neg\alpha]\!]^{M,e} \text{ iff False} \in [\![\alpha]\!]^{M,e} \\ &\text{False} \in [\![\neg\alpha]\!]^{M,e} \text{ iff True} \in [\![\alpha]\!]^{M,e}\end{aligned} \tag{2.4}$$

The semantic equations for the other logical connectives can be rewritten in a similar fashion, or, equivalently, they can be defined in terms of disjunction and negation. For example, define

$$\wedge = \lambda x, y.\ \neg(\neg x \vee \neg y)$$

or, if you prefer,

$$\begin{aligned}&\text{True} \in [\![\alpha \wedge \beta]\!]^{M,e} \text{ iff True} \in [\![\alpha]\!]^{M,e} \text{ and True} \in [\![\beta]\!]^{M,e} \\ &\text{False} \in [\![\alpha \wedge \beta]\!]^{M,e} \text{ iff False} \in [\![\alpha]\!]^{M,e} \text{ or False} \in [\![\beta]\!]^{M,e}\end{aligned} \tag{2.5}$$

Figure 2.1 displays the truth tables for negation, disjunction and conjunction in this three-valued logic. "T," "F," and "TF" abbreviate the names of the truth values in the obvious

[1] Accommodating value assignments at this point facilitates the incorporation of quantifiers in Chapter 4.

way.

It should now be clear that these semantic equations—(2.3), (2.4), and (2.5)—can be used to assign values to formulas in RP-models as well as in T-models. It should also be clear that T-models are precisely those RP-models where no atomic sentence is assigned {True, False}. Hence, as one would expect, each three-valued truth table contains the truth table for the two-valued Tarskian logic.

We say that model M *satisfies* sentence α if, and only if, $\text{True} \in [\![\alpha]\!]^{M,e}$; otherwise M *falsifies* α.[2] Furthermore, M satisfies a set of sentences iff it satisfies each sentence in the set and M falsifies a set of sentences iff it falsifies *some* sentence in the set. As usual, a set of sentences A RP-entails sentence β iff there is no model that satisfies A and falsifies β. A queried sentence of QFPC is retrievable from a KB of QFPC-sentences if, and only if, the queried sentence is RP-entailed by the sentences in the KB.

Let us now return to the example that motivated this definition of RP and ask, "Do P and $\neg P \vee Q$ RP-entail Q?" The answer is "no", because there are RP-models that satisfy both P and $\neg P$. For example, consider model M, which assigns {True, False} to P and {False} to Q. According to the definitions of the connectives, M assigns {True, False} to both $\neg P$ and $\neg P \vee Q$. So, M satisfies both P and $\neg P \vee Q$ but falsifies Q, and therefore is a countermodel to the claim that $P, \neg P \vee Q \models_{RP} Q$.

2.4. Properties of RP

Now that RP is defined, this section examines some of its properties. The preeminent results of this section show several logical equivalences for the RP logic and show that the retrievability relation that RP defines satisfies the Strength and the No-Chaining Criteria. These results lead directly to the decision procedure of the next section. However, before turning to them, we first observe some of the algebraic properties of this model theory. These algebraic properties prove very useful both in examining the properties of direct concern and in comparing this logic to others.

First note that the three truth values under the partial order of set inclusion form the upper semi-lattice shown in Figure 2.2. Because this semi-lattice forms three-fourths of Belnap's (1975; 1977) A4 lattice, I call it A3 and denote its partial order by $\sqsubseteq_{A3}$.

[2] It doesn't matter what e is since α is a sentence and hence contains no variables

$$\begin{tikzcd}[column sep=small]
 & \{\text{True, False}\} \arrow[dl, dash] \arrow[dr, dash] & \\
\{\text{True}\} & & \{\text{False}\}
\end{tikzcd}$$

Figure 2.2: The A3 Semi–Lattice

As usual, an n–ary logical connective can be viewed as a function from an n–tuple of truth values to a truth value—in this case $C{:}A3^n \rightarrow A3$. $\sqsubseteq_{A3}$ can be extended to order $A3^n$ on a point by point basis; i.e.,

$$\langle \alpha_1, \ldots, \alpha_n \rangle \sqsubseteq_{A3^n} \langle \beta_1, \ldots, \beta_n \rangle \text{ iff } \alpha_i \sqsubseteq_{A3} \beta_i \text{ for all } 1 \leq i \leq n.$$

Now, examination of the truth tables for disjunction and negation reveals that they are *monotonic* functions; that is, if $\vec{\alpha}, \vec{\beta} \in A3^n$ and C is the function denoted by "$\vee$" or "$\neg$" then $\vec{\alpha} \sqsubseteq_{A3^n} \vec{\beta}$ implies $C(\vec{\alpha}) \sqsubseteq_{A3} C(\vec{\beta})$. Considering equations (2.3) and (2.4), this observation is not surprising. The presence of True or False in the assignment to a formula is determined solely by the *presence, never the absence,* of True or False in the assignment to its subformulas. Finally, *all* logical connectives are monotonic, since they can be defined as compositions of disjunction and negation.

Further examination of the truth tables reveals that {True, False} is a fixed point for both negation and disjunction, and therefore a fixed point for all the logical connectives.

On its own, A3 is not very interesting. However, it can be used to construct complete upper semi–lattices of RP–models. This construction, and several others encountered in this thesis, require sets of models that are "compatible" in the following sense:

Definition: Compatible Set of Models
A set of models is *compatible* if, and only if, every model in the set has the same domain and each function symbol is assigned the same function in every model.

Several points deserve explicit mention: A set containing one model is compatible, as is the set of all Herbrand models. Compatible models interpret every term identically, though they may differ in their interpretations of the atomic sentences.

A maximal set of compatible RP–models forms a complete upper semi-lattice, called an A3S semi–lattice, under the partial ordering $\sqsubseteq_{A3S}$ defined as:

$$M \sqsubseteq_{A3S} M' \text{ iff } [\![\alpha]\!]^{M,e} \sqsubseteq_{A3} [\![\alpha]\!]^{M',e}, \tag{2.6}$$

for every atomic formula α and value assignment e

Here is a simple example of an A3S semi lattice. Consider the language whose lexicon has only three symbols: a zero-place function symbol, a, and two one place predicates, P and Q.

Every Herbrand model $\langle D, A\rangle$ for this language is such that $D=\{a\}$ and $A(a)=a$. The Herbrand models differ in their assignment of the truth values to $P(a)$ and $Q(a)$. Accordingly, a Herbrand model M can be described succinctly by the pair $\langle [\![P(a)]\!]^{M,e}, [\![Q(a)]\!]^{M,e}\rangle$. So, for example, $\langle TF, F\rangle$ is the Herbrand countermodel to the claim that $P(a), \neg P(a) \vee Q(a) \models_{RP} Q(a)$. Now, Figure 2.3 shows the A3S semi-lattice of the Herbrand models for this language. As our examination of the general properties of A3S semi-lattices continues, the reader may fi [illegible] useful to check all claims against this semi-lattice.

The minimal elements of an A3S semi-lattice are precisely the Tarskian models of the semi-lattice. Each A3S semi-lattice has a greatest element, $\top$, which assigns {True, False} to every atomic formula, regardless of the value assignment. Furthermore, because {True, False} is a fixed point of every logical connective, $\top$ assigns {True, False} to *every* formula. Therefore every sentence is RP-satisfiable. Here for the first time, a logical property has fallen out of the algebraic properties.

With A3 and A3S now in place, we are in a position to examine the connections between them. By the end of this section these connections will prove to be our most valuable tool in studying the properties of RP. The connections between a semi-lattice of models and a semi lattice of truth values are made by sentences. Associated with every sentence is a function, called the ***intension*** of the sentence, which maps each model to the value of the sentence in that model. Thus, the intension of sentence α, written $[\![\alpha]\!]$, is defined as

$$[\![\alpha]\!] \equiv_{def} \lambda M.\ [\![\alpha]\!]^{M} \tag{2.7}$$

This notion can be extended to formulas by speaking of the intension of a formula relative to a value assignment. Hence, if ψ is a formula and e is a value assignment, the intension of ψ relative to e, $[\![\psi]\!]^{e}$, is defined by

$$[\![\psi]\!]^{e} \equiv_{def} \lambda M.\ [\![\psi]\!]^{M,e} \tag{2.8}$$

The application of an intension, say $[\![\alpha]\!]^{e}$, to a model M is written as "$[\![\alpha]\!]^{M,e}$", not "$[\![\alpha]\!]^{e}(M)$".

Intensions have some important properties-- for example, the following simple one:

Monotonic Intension Lemma

Let ψ be any formula and e be any value assignment. Then $[\![\psi]\!]^{e}$ is monotonic i.e.,

$$M \sqsubseteq_{A3S} M' \text{ implies } [\![\psi]\!]^{M,e} \sqsubseteq_{A3} [\![\psi]\!]^{M',e}$$

Proof

Clearly the intension of every atomic formula is monotonic; that is the defining characteristic of $\sqsubseteq_{A3S}$ (see (2.6))! The logical connectives are also monotonic functions. Therefore the intension of a molecular formula, being composed of the intensions of atomic formulas and connectives, is also monotonic. ■

Consequently, if M satisfies a sentence so does every M' such that $M \sqsubseteq_{A3S} M'$. Therefore, a corollary to the above theorem is that every T-tautology is also an RP-tautology.

Every non-Tarskian model in an A3S semi-lattice is the least upper bound (l.u.b.) of a set of Tarskian models. If S is an A3S semi-lattice and $Y \subseteq$ A3S, then it is easy to see that the l.u.b. of Y is that model in S which assigns to each atomic sentence the union of the sentence's values in all the models of Y. Extending this to atomic formulas as well as atomic sentences, this observation is that for any atomic formula α and value assignment e,

$$[\![\alpha]\!]^{\sqcup Y,e} = \sqcup\{[\![\alpha]\!]^{y,e} \mid y \in Y\} \tag{2.9}$$

Any $[\![\alpha]\!]^e$—regardless of whether α is atomic—that satisfies this condition is said to be *completely additive.*[3] Complete additivity can be divided into two component properties: $[\![\alpha]\!]^e$ is completely additive iff it is both

completely t-additive: $\text{True} \in [\![\alpha]\!]^{\sqcup Y,e}$ iff $\text{True} \in \sqcup\{[\![\alpha]\!]^{y,e} \mid y \in Y\}$,
for every Y that is a subset of some A3S lattice, and

completely f-additive: $\text{False} \in [\![\alpha]\!]^{\sqcup Y,e}$ iff $\text{False} \in \sqcup\{[\![\alpha]\!]^{y,e} \mid y \in Y\}$,
for every Y that is a subset of some A3S lattice.

The following lemma characterizes some formulas whose intensions are completely additive.

Complete Additivity Lemma

Let α and β be formulas and e be a value assignment. Then:

(1) If α is atomic then $[\![\alpha]\!]^e$ is completely additive.
(2) If $[\![\alpha]\!]^e$ is completely t-additive then $[\![\neg\alpha]\!]^e$ is completely f additive.
(3) If $[\![\alpha]\!]^e$ is completely f-additive then $[\![\neg\alpha]\!]^e$ is completely t-additive.
(4) If $[\![\alpha]\!]^e$ and $[\![\beta]\!]^e$ are completely t-additive then so is $[\![\alpha\vee\beta]\!]^e$.
(5) If $[\![\alpha]\!]^e$ and $[\![\beta]\!]^e$ are completely f-additive then so is $[\![\alpha\wedge\beta]\!]^e$.

Proof

(1) Obvious. Previously stated as (2.9).

(2) $\text{False} \in [\![\neg\alpha]\!]^{\sqcup Y,e}$ iff $\text{True} \in [\![\alpha]\!]^{\sqcup Y,e}$ (def. of $\neg$)
iff $\text{True} \in \sqcup\{[\![\alpha]\!]^{y,e} \mid y \in Y\}$ (assumption)
iff for some $y \in Y$, $\text{True} \in [\![\alpha]\!]^{y,e}$ (def. of A3)
iff for some $y \in Y$, $\text{False} \in [\![\neg\alpha]\!]^{y,e}$ (def. of $\neg$)
iff $\text{False} \in \sqcup\{[\![\neg\alpha]\!]^{y,e} \mid y \in Y\}$ (def. of A3)

(3) Similar to (2).

(4) $\text{True} \in [\![\alpha\vee\beta]\!]^{\sqcup Y,e}$
iff $\text{True} \in [\![\alpha]\!]^{\sqcup Y,e}$ or $\text{True} \in [\![\beta]\!]^{\sqcup Y,e}$ (def. of $\vee$)

[3] This term is used by some authors (e.g., Sanderson (1973)) while others use "meet homomorphic" or "upper homomorphic."

iff True $\in \sqcup\{[\![\alpha]\!]^{y,e} | y \in Y\}$ or True $\in \sqcup\{[\![\beta]\!]^{y,e} | y \in Y\}$ (assumption)

iff for some $y \in Y$, True $\in [\![\alpha]\!]^{y,e}$, or some $y \in Y$, True $\in [\![\beta]\!]^{y,e}$ (def. of A3)

iff for some $y \in Y$, True $\in [\![\alpha]\!]^{y,e}$ or True $\in [\![\beta]\!]^{y,e}$ (metalogical)

iff for some $y \in Y$, True $\in [\![\alpha \vee \beta]\!]^{y}$ (def. of $\vee$)

iff True $\in \sqcup\{[\![\alpha \vee \beta]\!]^{y,e} | y \in Y\}$ (def. of A3)

(5) Similar to (4). ■

Not every intension is completely additive. For example, one can observe that the intension of $P \wedge Q$ is not completely t-additive by considering the simple case in which:

$$Y = \{M_1, M_2\}$$

$$[\![P]\!]^{M_1,e} = \text{True} \qquad [\![Q]\!]^{M_1,e} = \text{False}$$

$$[\![P]\!]^{M_2,e} = \text{False} \qquad [\![Q]\!]^{M_2,e} = \text{True}$$

Consequently,

$$[\![P]\!]^{\sqcup Y,e} = \{\text{True}, \text{False}\} \qquad [\![Q]\!]^{\sqcup Y,e} = \{\text{True}, \text{False}\}$$

$$[\![P \wedge Q]\!]^{\sqcup Y,e} = \{\text{True}, \text{False}\}$$

$$\sqcup\{[\![P \wedge Q]\!]^{y,e} | y \in Y\} = \text{False}$$

So, $[\![P \wedge Q]\!]^{\sqcup Y,e}$ contains True but $\sqcup\{[\![P \wedge Q]\!]^{y,e} | y \in Y\}$ does not, thereby demonstrating that the intension of $P \wedge Q$ is not completely t-additive.[4]

Fact Intension Theorem

Every fact has a completely t-additive intension.

Proof

The Theorem follows immediately from the Complete Additivity Lemma. By statement (1), the intension of every atomic formula is completely additive, and thus, by (2) and (3), so is the intension of every literal. Therefore, by (4), a disjunction of literals must have a completely t-additive intension. ■

We now turn directly to the logical properties of RP. As in two-valued logic, it is often convenient to restrict our attention to the Herbrand models. This selective attention is licensed by the Herbrand-Model Lemma, which says that for certain purposes the Herbrand models are representative of all models. This lemma holds in RP for precisely the same rea-

[4] Notice that Y is not a directed set so this example does not demonstrate that the intension of $P \wedge Q$ is not continuous

sons that it holds in a two-valued logic, so it is stated here without proof.[5]

Herbrand-Model Lemma
Let A and B each be a set of sentences. If no Herbrand RP-model satisfies A and falsifies B then $A \models_{RP} B$.[6]

Since the set of all Herbrand models is compatible, it forms an A3S semi-lattice that, for certain purposes, is representative of all A3S semi-lattices. Thus, for example, one can examine the semi-lattice of Figure 2.3 in order to verify that $\emptyset \models_{RP} P(a) \vee \neg P(a)$ and that $P(a) \models_{RP} P(a) \vee Q(a)$.

It is now easy to prove that RP meets the No-Chaining and Strength Criteria.

No-Chaining Theorem
Let Φ be a set of sentences and α be a fact. Then $\Phi \models_{RP} \alpha$ iff for some $\phi \in \Phi$ $\phi \models_{RP} \alpha$.

Proof
If clause: Obvious.
Only-if clause: Assuming that there is no $\phi \in \Phi$ such that $\phi \models_{RP} \alpha$, I construct a model M^* that satisfies every $\phi \in \Phi$ but falsifies α. From the assumption it follows that for every $\phi \in \Phi$ there is a Herbrand RP-model, M_ϕ that satisfies ϕ and falsifies α. Let $M^* = \bigsqcup\{M_\phi | \phi \in \Phi\}$. Now, for every $\phi \in \Phi$, $M_\phi \sqsubseteq_{A3S} M^*$ and therefore, by the Monotonic Intension Lemma, M^* satisfies ϕ. Moreover, True $\notin \bigsqcup\{[\![\alpha]\!]^{M_\phi} | \phi \in \Phi\}$ because each M_ϕ falsifies α; therefore, since α is a fact, the Fact Intension Theorem assures us that True $\notin [\![\alpha]\!]^{M^*}$. ■

From here, the presentation could proceed in two ways, each building on the T-Decision Theorem for Facts. One approach would proceed by first proving the RP-Decision Theorem for Facts, which says that RP-entailment between facts can be decided in the same way that the T-Decision Theorem for Facts says to decide T-entailment. The Strength Theorem would then follow immediately. The other approach would begin by proving the Strength Theorem, which then yields the RP-Decision Theorem for Facts as an immediate consequence. However, since both the RP-Decision Theorem for Facts and the Strength Theorem are interesting in their own right, and since both have simple proofs, I take a third approach by proving each theorem independently.

Strength Theorem
If ϕ is a fact and ψ is a sentence then $\phi \models_{RP} \psi$ iff $\phi \models_{T} \psi$.

[5] Robinson (1979) gives an excellent exposition of Herbrand models

[6] Because this is the propositional case, Skolem Normal Form is not an issue

Proof

Only-if clause: Immediate since every T-model is an RP-model.

If clause: Assuming $\phi \models_T \psi$ and that M is an RP-model satisfying ϕ, I show that M satisfies ψ. Since M is the l.u.b. of a set of T-models, the Fact Intension Theorem implies that there must be some T-model $M' \sqsubseteq_{A3S} M$ that satisfies ϕ. Furthermore, since $\phi \models_T \psi$, M' must also satisfy ψ. Therefore, by the Monotonic Intension Lemma, M must satisfy ψ. ■

RP-Decision Theorem for Facts

For any facts, ϕ_1 and ϕ_2, $\phi_1 \models_{RP} \phi_2$ iff either

(1) ϕ_2 has complementary literals, or

(2) $LITERALS(\phi_1) \subseteq LITERALS(\phi_2)$.

Proof

If clause: Assume Condition (2). From the definition of $\vee$ observe that an RP-model satisfies a disjunction iff it satisfies one of the disjuncts. Therefore, any RP-model satisfying ϕ_1 also satisfies ϕ_2; i.e., $\phi_1 \models_{RP} \phi_2$. On the other hand, assume Condition (1). From the RP-truth table for $\neg$ observe that each literal or its complement is satisfied by any RP-model. Thus, (by the argument above) any disjunction containing complementary literals is satisfied by every model, and therefore $\phi_1 \models_{RP} \phi_2$.

Only-if clause: Immediate since every T-model is an RP-model. ■

Notice that this theorem and the proof of its if-clause are identical to the T-Decision Theorem for Facts and the proof of its if-clause (except of course that one concerns T and the other RP). Moreover, the only-if-clause of this theorem could be proved by mimicking the proof of the only-if-clause of the T-Decision Theorem for Facts.

This section concludes by presenting some RP-equivalences. However, we first examine the notion of equivalence itself, which must be handled with care when working with a multi-valued logic.

In general, two notions can be distinguished: mutual entailment ($\cong$), defined in (2.10), and equivalence ($\equiv$), defined in (2.11).

$$\alpha \cong \beta \text{ iff } \alpha \models \beta \text{ and } \beta \models \alpha \tag{2.10}$$

$$\alpha \equiv \beta \text{ iff } [\![\alpha]\!]^{M,e} = [\![\beta]\!]^{M,e}, \text{ for all } M \text{ and } e. \tag{2.11}$$

The relationship between these two definitions becomes clear if they are rewritten as

$$\alpha \cong \beta \text{ iff for all } M \text{ and } e,\ \text{True} \in [\![\alpha]\!]^{M,e} \text{ iff True} \in [\![\beta]\!]^{M,e} \tag{2.10$'$}$$

$$\alpha \equiv \beta \text{ iff for all } M \text{ and } e,\ \text{True} \in [\![\alpha]\!]^{M,e} \text{ iff True} \in [\![\beta]\!]^{M,e}, \text{ and} \quad \text{False} \in [\![\alpha]\!]^{M,e} \text{ iff False} \in [\![\beta]\!]^{M,e}. \tag{2.11$'$}$$

Notice that (2.10′) defines equivalence over all formulas whereas (2.10) only defines it over the sentences.

Trivially in T, and non-trivially in some multi-valued logics (Belnap, 1975; 1977), these two definitions coincide. However, in RP they do not. This is demonstrated by the sentences $P \vee \neg P$ and $Q \vee \neg Q$, which (mutually) RP-entail each other, but are not RP-equivalent. Since each of these two sentences is an RP-tautology, every model assigns each of them a value containing True. However, in a model that assigns {True, False} to P and {True} to Q, $P \vee \neg P$ is assigned {True, False} while $Q \vee \neg Q$ is assigned {True}.

The choice of whether to use mutual entailment or equivalence depends on one's purposes. This thesis is often concerned with substituting equals for equals and is therefore concerned with the notion of equivalence.

RP-Equivalence Theorem
If α, β, and ψ are formulas of QFPC then

(1) $\alpha \vee \alpha \equiv_{RP} \alpha$

(2) $\alpha \wedge \alpha \equiv_{RP} \alpha$

(3) $\neg\neg\alpha \equiv_{RP} \alpha$

(4) $\alpha \wedge (\beta \vee \psi) \equiv_{RP} (\alpha \wedge \beta) \vee (\alpha \wedge \psi)$

(5) $\alpha \vee (\beta \wedge \psi) \equiv_{RP} (\alpha \vee \beta) \wedge (\alpha \vee \psi)$

(6) $\alpha \vee \beta \equiv_{RP} \neg(\neg\alpha \wedge \neg\beta)$

(7) $\alpha \wedge \beta \equiv_{RP} \neg(\neg\alpha \vee \neg\beta)$

Proof
Construct the truth tables. ■

2.5. The Retrieval Algorithm

In certain cases it is simple to see how retrievability can be decided. The Fact Decision Theorem for RP provides a method for deciding whether or not a query consisting of a single fact is retrievable from a KB containing a single fact. The No Chaining Theorem extends the method to the case where the KB contains any finite number of facts. Finally, the semantics of conjunction, which says that an RP-model satisfies a conjunction iff it satisfies each conjunct, provides for the case where the KB contains conjunctions of facts and the query is a conjunction of facts. Hence, retrieval problems based on conjunctions of facts play a special role in the analysis.

Formulas that are conjunctions of disjunctions of literals are traditionally said to be in *conjunctive normal form* (CNF). The use of CNF in the decision procedure divides the remainder of this section naturally into two subsections. The first is concerned with transforming arbitrary QFPC formulas into CNF while the second is concerned with the details of deciding if a CNF query is retrievable from an KB of CNF sentences.

The sections and chapters that follow use some terminology, which is now introduced. Since every retrieval problem is characterized by the set of sentences in the KB and the queried sentence, I introduce into the meta-language objects, called *sequents,* that encode such characterizations. A sequent is, quite simply, a pair whose first element is a set of sentences and whose second is a single sentence. According to this definition a sequent, like a pair of numerals, is merely a syntactic object; it has no assertional import; it is neither true nor false. A sequent composed of kb and q is written as "$kb \Rightarrow q$"; kb is called the *antecedent* of the sequent, and q is called the *succedent* of the sequent. In writing a sequent such as $\{P(a), Q(a)\} \Rightarrow R(a)$ I usually omit the set signs and simply write $P(a), Q(a) \Rightarrow R(a)$.

A sequent is said to be finite or infinite according to the number of sentences that it contains. Both finite and infinite sequents are used frequently throughout the remainder of this thesis though, of course, retrieval problems always correspond to finite sequents. This chapter is concerned only with sequents of QFPC, i.e., sequents containing only sentences of QFPC. When we examine retrievers operating on other languages, sequents of those languages will be used.

Finally, I stretch the terminology slightly and say that a sequent $kb \Rightarrow q$ is in an entailment relation, by which I mean that kb entails q. Similarly, I say that a sequent is in a retrievability or a provability relation.

2.5.1. The Conjunctive Normal Transformation

The *conjunctive normal transformation* (CNT) is a well-known algorithm for converting any QFPC formula to a T-equivalent CNF QFPC formula. After this transformation is presented, a theorem will state that the transformation's output is also RP-equivalent to its input.

Definition: Conjunctive Normal Transformation
Input: An arbitrary formula of QFPC.
Output: A CNF formula of QFPC.

(1) Eliminate all occurrences of $\rightarrow$ and $\leftrightarrow$ by the following rules:
  Rewrite $\psi \rightarrow \phi$ to $\neg\psi \vee \phi$
  Rewrite $\psi \leftrightarrow \phi$ to $(\neg\psi \wedge \neg\phi) \vee (\psi \wedge \phi)$

(2) By the following rules push all occurrences of $\neg$ inward so that only atomic sentences are negated:
  Rewrite $\neg\neg\psi$ to ψ
  Rewrite $\neg(\psi_1 \vee \cdots \vee \psi_n)$ to $\neg\psi_1 \wedge \cdots \wedge \neg\psi_n$
  Rewrite $\neg(\psi_1 \wedge \cdots \wedge \psi_n)$ to $\neg\psi_1 \vee \cdots \vee \neg\psi_n$

(3) With the following rule, distribute $\vee$ over $\wedge$ so that no $\wedge$ occurs within the scope of an $\vee$:
  Rewrite $\psi \vee (\phi_1 \wedge \cdots \wedge \phi_n)$ to $(\psi \vee \phi_1) \wedge \cdots \wedge (\psi \vee \phi_n)$

Conjunctive–Normal–Transformation Theorem
Every QFPC formula is RP–equivalent to its conjunctive normal transform.

Proof
Each of the rules rewrites a formula to an RP–equivalent formula; hence the transformation as a whole rewrites sentences to RP–equivalent sentences. The RP–equivalences that justify the rewrite rules of step (1) follow from the definitions of $\rightarrow$ and $\leftrightarrow$, while the others are from the RP–Equivalence Theorem. ∎

A sequent is in CNF iff all of its sentences are. The conjunctive normal transformation can be used to put arbitrary sequents into CNF merely by replacing every formula in the sequent with its transform. Because this operation replaces equals with equals, the resulting sequent is in the RP–entailment relation iff the original one is. More concisely, I say that the CNT *preserves* RP–entailment.

2.5.2. Retrievability of CNF Sequents

As stated at the outset of Section 2.5, the RP–Decidability Theorem for Facts, the No–Chaining Theorem and the semantics of conjunction make clear how to decide $\models_{RP}$ for CNF sequents of QFPC. Nevertheless this section presents a decision algorithm, not because it is of intrinsic interest, but because it lays the foundation for the more-interesting algorithms of the chapters that follow. The algorithm is called the "Ground Sequent Retrieval Algorithm" or simply the "GSRA"; the word "ground" is used to distinguish this algorithm, which operates on the sentences of QFPC, from the algorithm of the next section, which operates on sentence schemas.

Ground Sequent Retrieval Algorithm

Input: $kb \Rightarrow q$, a CNF sequent of QFPC.

Output: SUCCESS or FAILURE

```
(1)  let $s$ = number of conjuncts in $q$
(2)  let $q_i$ = $i^{th}$ conjunct of $q$ $(1 \leq i \leq s)$
(3)  let $K$ be the set containing every conjunct of every sentence in $kb$
(4)  for $i = 1$ to $s$ do
(5)      choose to do either step A) or step B)
(6)      A) choose $p_i$, a positive literal in $q_i$
(7)         choose $n_i$, the complement of a negative literal in $q_i$
(8)         if $n_i = p_i$
(9)         then continue
(10)        else FAIL
(11)     B) choose $b_i \in K$
(12)        let $l_{i,1}, \ldots, l_{i,m}$ be the literals of $b_i$
(13)        choose $F_i$, a total function from $LITERALS(b_i)$ to $LITERALS(q_i)$
(14)        if $\langle l_{i,1}, \ldots, l_{i,m} \rangle = \langle F_i(l_{i,1}), \ldots, F_i(l_{i,m}) \rangle$
(15)        then continue
(16)        else FAIL
(17) SUCCEED
```

The "choose" statements in this algorithm make non-deterministic choices, which means that the algorithm succeeds iff there is *some* sequence of choices that leads to the "SUCCEED" statement at the end. If a choice must be made from an empty set of options then the execution fails. This can happen on line (6) if q_i does not contain both a positive and a negative literal, or on line (11) if $K = \emptyset$.

This interpretation of the choose statements motivates the following definition of provability. We shortly shall see the justification for using the term "provable."

Definition: GSRA-Provability

Let $kb \Rightarrow q$ be a CNF sequent of QFPC. Then, q is GSRA-provable from kb (written $kb \vdash_{GSRA} q$) iff there is some sequence of choices for which the Ground Sequent Retrieval Algorithm halts with SUCCESS when input $kb \Rightarrow q$.

The following theorem states that the GSRA meets the retrieval specification of Section 2.3; that is, RP-entailment and GSRA-provability are one and the same. Armed with the results of Section 2.4, the proof of this theorem is straightforward.

GSRA Correctness Theorem
Let $kb \Rightarrow q$ be a CNF sequent of QFPC. Then, $kb \models_{RP} q$ iff $kb \vdash_{GSRA} q$.

Proof
The GSRA succeeds iff every iteration of the loop (with i ranging from 1 to s) succeeds. The i^{th} iteration succeeds iff either step (A) or step (B) succeeds. Step (A) succeeds iff q_i has complementary literals and step (B) succeeds iff for some $b_i \in K$, $LITERALS(b_i) \subseteq LITERALS(q_i)$. Thus, by the Fact Decision Theorem for RP, the i^{th} iteration succeeds iff $b_i \models_{RP} q_i$ for some $b_i \in K$. By the No-Chaining Theorem this happens precisely when $K \models_{RP} q_i$. The semantics of $\wedge$ says that a conjunction is satisfied iff each of its conjuncts is. Therefore, $K \models_{RP} q_i$, for $1 \leq i \leq s$, iff $K \models_{RP} q$ iff $kb \models_{RP} q$. ■

Not only is the GSRA correct, but when executed with a finite input each of its steps is effectively computable and the algorithm terminates. Crucial to this claim is that, when working with a finite input, every non-deterministic choice is made from a finite set of options that can be effectively constructed. For instance, in choosing an F_i in line (12) the set of all total functions from $LITERALS(b_i)$ to $LITERALS(q_i)$ can be constructed. If b_i has r literals and q_i has s literals then there are s^r such functions, each of which can be finitely represented. Because all choices are made from finite sets a deterministic machine can execute this non-deterministic algorithm simply by trying all combinations of choices.

A survey of all combinations of choices is, of course, a search space. A search space can be displayed as a tree in which each node represents a choice that must be made and the arcs emanating from a node represent all of the options available and are labeled as such. The node at the end of an arc represents the next choice that must be made after that arc is chosen. The first choice that an execution of the GSRA encounters is located at the root of the tree. Every path originating from the root represents an initial sequence of choices that an execution could make. Hence, a path from the root to a leaf represents the sequence of choices made during some complete execution of the program and the leaf is labeled "SUCCEED" or "FAIL" according to the outcome of that execution.

Figure 2.4 displays the search space implicitly defined when the GSRA is executed on the input sequent $P, R \Rightarrow (P \vee Q) \wedge (R \vee \neg R)$. Because some of the leaf nodes are labeled "SUCCEED", the algorithm successfully retrieves $(P \vee Q) \wedge (R \vee \neg R)$ form the KB containing P and R.

A consequence of the previous discussion is that every finite sequent has a finite search space. In attempting to retrieve a sentence with s conjuncts the GSRA iterates at most s times, making three choices on each iteration. Hence, a search space has a depth of at most $3s$. Furthermore, the tree is finitary since all choices are made from a finite number of options.

Search spaces in the style of Figure 2.4 can be summarized in another style of display. Observe that Subtree I in Figure 2.4 displays the following: an unsuccessful attempt to show that $P \vee Q$ has complementary literals (terminating at node [2]), two unsuccessful attempts to show $\{R\} \subseteq \{P, Q\}$ (terminating at nodes [8] and [9]), an unsuccessful attempt to show $\{P\} \subseteq \{P, Q\}$ (terminating at node [7]), and a successful attempt to show $\{P\} \subseteq \{P, Q\}$. Each of the unsuccessful attempts corresponds to an unsuccessful attempt to apply an inference rule in a certain manner. Traditionally, deductive search spaces display only successful rule applications, suppressing the failed attempts. Of course, a successful rule application is not necessarily part of a successful proof.

From here on I follow this lead by displaying only those choices that allow execution to continue to the end of the iteration in which they occur. So, only that part of Subtree I containing nodes [1], [3], [4] and [6] is of interest. Since this represents the three choices made on a particular iteration, they are collapsed into a single arc labeled with the three choices. Following this convention, Figure 2.4 can be redisplayed as Figure 2.5. Notice that arcs 1–6, 6–11, and 6–15 of Figure 2.5 summarize respectively Subtrees I, II, and III of Figure 2.4.

The discussion has brought us to an appropriate point for making a rare comment about implementation issues. A good implementation of the GSRA could indeed avoid many of the unsuccessful choices that are suppressed in the condensed style of search space. For example, in step (B), b_i is chosen and then a test is made to see if $LITERALS(b_i) \subseteq LITERALS(q_i)$. An implementation could exploit an indexing scheme for accessing all elements of K that pass the subset test while avoiding those that fail.

This chapter has progressed from a model theory directly to an algorithm, avoiding proof theory in its traditional form of axioms and inference rules. This is appropriate in a study of knowledge retrieval, which is, after all, a computational process. The simplicity of RP allowed for a smooth transition from the model theory to the algorithm.

Though this study has no need for a traditional proof theory, proofs themselves are useful objects. While the retrieval algorithm can determine what follows from what, a proof provides an argument that it does follow. Furthermore it is often valuable to examine the properties of proofs, such as their size, and relationships between proofs, such as the so-called lifting relationship, which is used in the next chapter. Fortunately, we already have at hand constructions that can naturally be called proofs; a proof is a path through a search space originating at the root and terminating at a leaf node labeled "SUCCEED". This definition is appropriate because a claim that a certain path is a proof of a certain sequent can be readily tested.

One last issue deserves attention. Recall that the GSRA works for any query q in CNF; in particular, it works for any query obtained by permuting the conjuncts of q. Many problem solvers, including theorem-provers, use a *selection function* to determine what order to work on the components of a conjunctive goal. The GSRA could employ a selection function.

At the outset of each iteration it would select one conjunct of q from those that have not been selected previously. It is clear that the correctness of the GSRA is independent of what selection function is used.

Though the order in which the conjuncts of a query are retrieved does not affect the correctness of the GSRA, it can affect the search space. The obvious example is of a query that fails. The sooner that a failing conjunct is selected, the smaller the resulting search space. Figure 2.6 shows the search space for $Q \Rightarrow (P \vee Q \vee \neg P) \wedge \neg Q$ while the search space for $Q \Rightarrow \neg Q \wedge (P \vee Q \vee \neg P)$ consists of a single node with no arcs.

Even for successful queries the order matters. By reordering $P, R \Rightarrow (P \vee Q) \wedge (R \vee \neg R)$ to $P, R \Rightarrow (R \vee \neg R) \wedge (P \vee Q)$ the search space of Figure 2.5 is transformed to that of Figure 2.7. The modifications caused by reordering successful queries are inconsequential because, at least presently, we can be content with finding one proof, a task whose difficulty is independent of goal ordering.

Not only can the conjuncts of a query be reordered, but they can be retrieved independently of each other. The latter is a stronger property that entails the first. The stronger property obtains as a consequence of the semantic definition of conjunction, which implies that

$$kb \models_{RP} q_1 \wedge \cdots \wedge q_n \text{ iff } kb \models_{RP} q_1 \text{ and } \cdots \text{ and } kb \models_{RP} q_n$$

for any sequent $kb \Rightarrow q_1 \wedge \cdots \wedge q_n$. Furthermore, if this sequent happens to be in CNF, then the GSRA Correctness Theorem implies that

$$kb \vdash_{\overline{GSRA}} q_1 \wedge \cdots \wedge q_n \text{ iff } kb \vdash_{\overline{GSRA}} q_1 \text{ and } \cdots \text{ and } kb \vdash_{\overline{GSRA}} q_n$$

Alternatively, this property can be observed by direct examination of the GSRA. Each iteration of the algorithm is independent of all previous iterations; nothing computed during one iteration is carried forward for use during future iterations. Problems that can be divided into independent subproblems the way that CNF retrievability can are called "decomposable" and an algorithm that decomposes such problems confronts an AND/OR search space (Nilsson, 1980).

Of the two properties, selection-function independence and decomposability, the latter is more important in constructing a retriever for QFPC sequents that is efficient. Yet this chapter now concludes having examined selection functions more closely than decomposition. This has been done in an attempt to lay the groundwork for examining the retrievers of the following chapters, all of which are independent of selection function though none are decomposable.

¬	
T	F
F	T
TF	TF

∧	T	F	TF
T	T	F	TF
F	F	F	F
TF	TF	F	TF

∨	T	F	TF
T	T	T	T
F	T	F	TF
TF	T	TF	TF

Figure 2.1: Truth Tables for RP

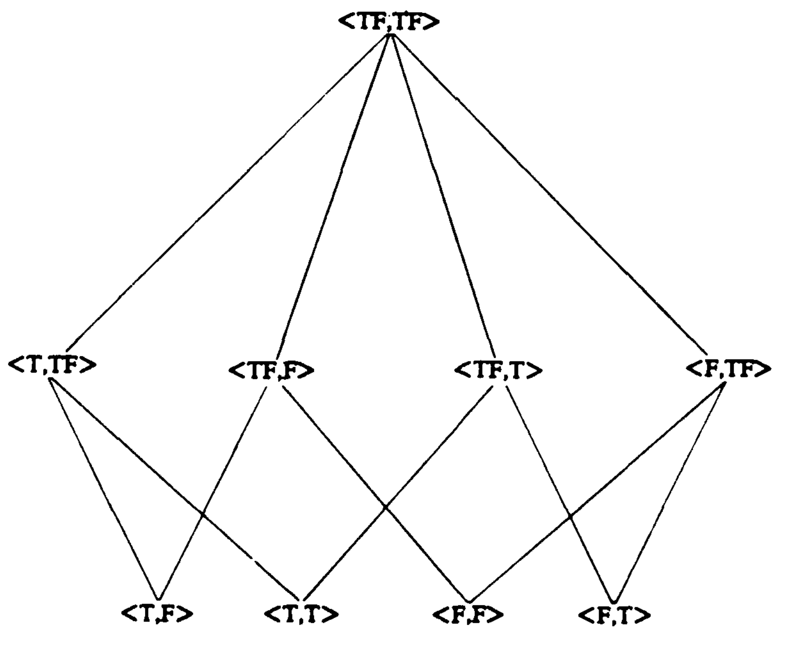


Figure 2.3: A Herbrand A3S Semi-Lattice

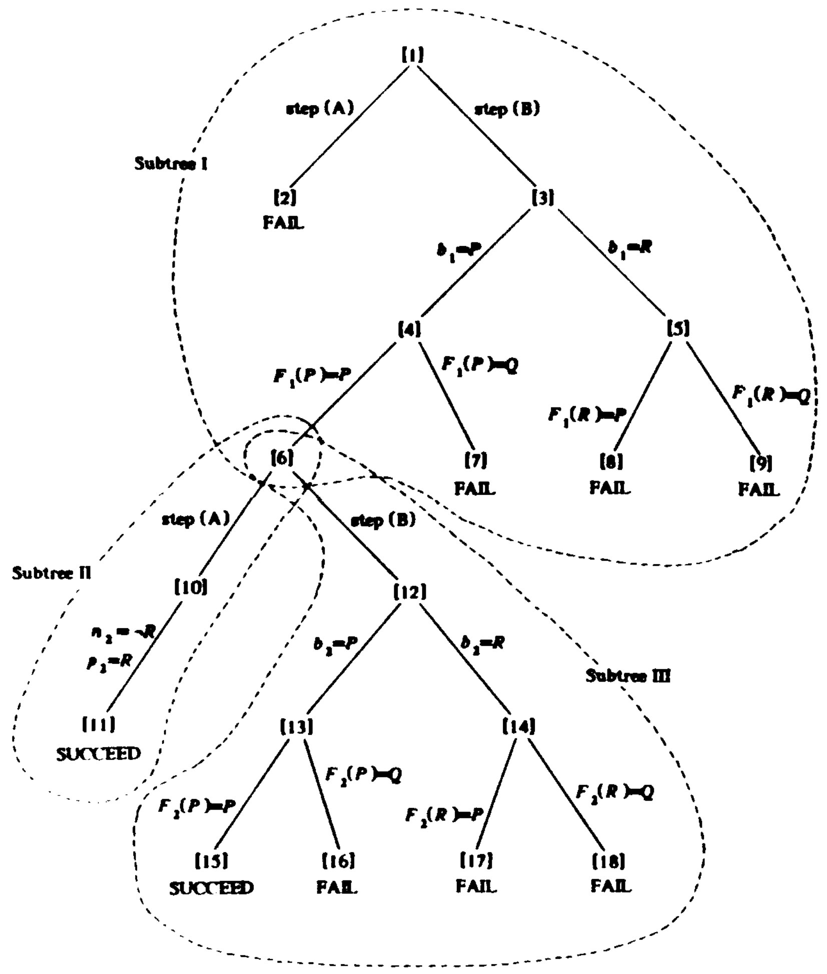


Figure 2.4: The Search Space of P, R $\Rightarrow (P \vee Q) \wedge (R \vee \neg R)$

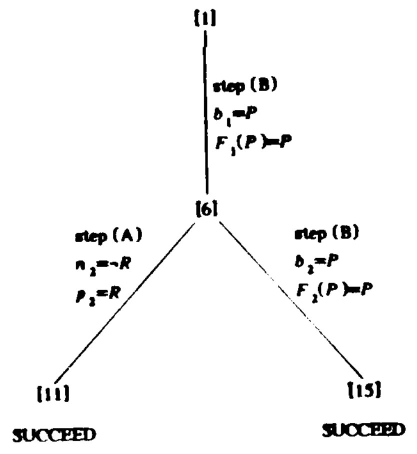


Figure 2.5: The Condensed Search Space of P,R $\Rightarrow (P \vee Q) \wedge (R \vee \neg R)$

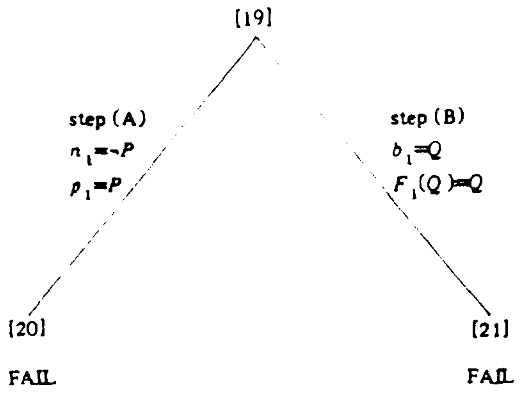


Figure 2.6: The Search Space for Q ⇒ (P ∨ Q ∨ ¬P) ∧ ¬Q

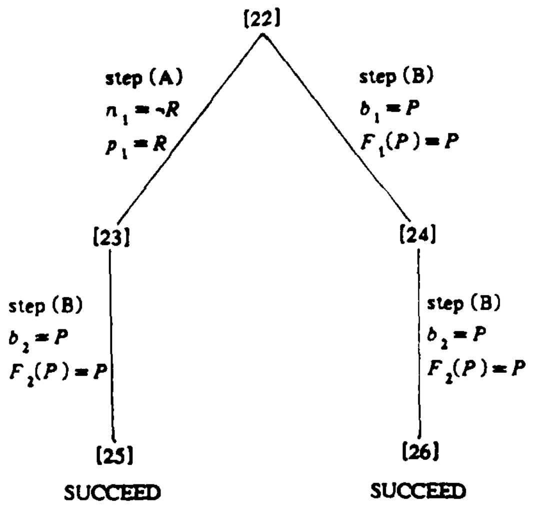


Figure 2.7: The Search Space of P, R ⇒ (R ∨ ¬R) ∧ (P ∨ Q)

Chapter 3

A Retriever that Handles Wh-Queries

The retriever specified in the previous chapter responds to queries merely by indicating success or failure. In this sense, such queries correspond to yes/no questions in English. Accessing a KB with this retriever would be like playing a game of Twenty Questions. This chapter specifies a retriever that responds to certain queries by supplying a set of answers. Such queries are more like English wh-questions.

The specification of this retriever generalizes the specification of the last chapter by allowing *sentence schemas* of QFPC as queries and as elements of the KB. Thus, we are now concerned with retrieval problems that are characterized by finite schematic sequents of QFPC. This chapter defines what schematic sentences are and specifies a retrievability relation for schematic sequents in terms of the RP-entailment relation for non-schematic sequents.

3.1. An Approach to Wh-Queries

Let us consider the rather obvious way that the retriever of the last chapter can be used in answering a yes/no question. Then by analogy we can reason about what capabilities a retriever would need in order for it to be used in answering a wh-question. Though the analogy proceeds by considering a sequence of English sentences, I am *not* making any linguistic claims.

Suppose we wished to answer the question

Does Roth sell expert systems? (3.1)

Ideally we would like to turn to the retriever and issue the imperative

Tell me whether "Roth sells expert systems" is true. (3.2)

but, of course, the best that the retriever can do is respond to

Tell me whether "Roth sells expert systems" is retrievable. (3.3)

Indeed, this is what the retriever does when given the query "Roth sells expert systems" (expressed in the logical language, of course). In general, we give the retriever a queried sentence q and in doing so issue the command

Tell me whether q is retrievable. (3.4)

Now let us consider wh-queries in an analogous way. Suppose that we wish to answer the question

Who sells what? (3.5)

which corresponds to the imperative

Name every x and y such that x sells y. (3.6)

However, the form of this imperative bears little resemblance to the form of its analog, (3.2). The former imperative contains a quoted sentence and asks for a report on its truth. Putting (3.6) in this form results in

Name every pair of terms such that "x sells y" is a true sentence when the first element of the pair is substituted for x and the second element of the pair is substituted for y. (3.7)

The awkwardness of this request is caused by the need to correlate the entries in the requested pairs with x and y. Some of this can be avoided by asking for a substitution rather than a pair. That is, instead of asking for pairs such as ⟨Roth, expert systems⟩ and ⟨Wilson, designer drugs⟩ it is simpler to ask for substitutions such as "substitute 'Roth' for x and 'expert systems' for y" and "substitute 'Wilson' for x and 'designer drugs' for y". By asking for substitutions, the answer itself supplies the necessary correlations. Following this strategy, (3.7) can be rephrased as

Name every substitution for x and y whose application to "x sells y" results in a true sentence. (3.8)

However, as before, the best that the retriever can do is respond to the command

Name every substitution for x and y whose application to "x sells y" results in a retrievable sentence. (3.9)

In general, we give the retriever a sentence schema q and in doing so issue the command

Name every substitution for the schematic variables in q whose application to q results in a retrievable sentence. (3.10)

This all but specifies a retrievability relation that handles wh-queries in the form of schematic queries. In (3.10) the meaning of "retrievable" is precisely that given by the retrievability relation $\models_{RP}$. So, the specification of retrievability formalized in this chapter does not fold, spindle or mutilate RP—rather it defines retrievability of schematic queries in terms of substitution and RP-entailment.

Following the next section's formalization of the notion of substitution, Section 3.3 specifies the retrievability relation and Section 3.4 presents a retrieval algorithm that meets the specification. As the reader familiar with automated deduction might suspect, this algorithm uses unification to *lift* the Ground Sequent Retrieval Algorithm to the schematic level.

3.2. Substitutions and Unifiers

This section presents the basic and fairly standard definitions and theorems concerning substitutions and unifiers. The reader familiar with this subject may wish to skim the section to familiarize himself with my notation. I have attempted to include those and only those definitions and theorems on which the remainder of this chapter rests. Plotkin (1972), Robinson (1979), Huet and Oppen (1980), and Eder (1985) cover this topic in more detail.

Intuitively, a substitution is a function that maps an expression such as "x sells y" to another expression such as "Wilson sells designer drugs." Basically, the mapping works by replacing variables with expressions while leaving everything else alone.

I follow the common conventions of naming substitutions by Greek letters (in this case, θ, σ, γ, ρ, λ and Θ) and writing the application of a substitution θ to an expression e as $e\theta$ rather than $\theta(e)$.

Definition: Substitution

A ***substitution*** is a function θ from expressions to expressions such that for every expression e:

(1) If e is a constant then $e\theta = e$.

(2) If e is composed of $e_1, e_2, \ldots, e_n$ then $e\theta$ is composed of $e_1\theta, e_2\theta, \ldots, e_n\theta$ in the same manner.

(3) If e is a variable then $e\theta$ is an expression.

A substitution may also be applied to a set or a tuple of expressions; in such a case the substitution is merely applied to each expression in the set or tuple. Thus, if E is a set of expressions, then $E\theta = \{e\theta | e \in E\}$ and if $\vec{e} = \langle e_1, e_2, \ldots, e_n \rangle$ is an n-tuple of expressions, then $\vec{e}\theta = \langle e_1\theta, e_2\theta, \ldots, e_n\theta \rangle$.

Observe that substitutions are uniquely determined by their treatment of the variables; that is, $\theta_1 = \theta_2$ iff $v\theta_1 = v\theta_2$ for every variable v. The substitutions that arise in this work are such that $v\theta \neq v$ for only a finite number of variables v. By an abuse of terminology, the set of all such variables is called the ***domain*** of θ, or simply $DOM(\theta)$. θ is said to be a substitution *for* a set of variables V if $DOM(\theta) \subseteq V$.

$VARS(e)$ denotes the set of all variables that occur in expression e. If it is empty, e is said to be ***ground***. A substitution is said to be ground if it maps every variable in its domain to a ground expression.

e' is said to be an ***instance*** of e if $e' = e\theta$, for some substitution θ. Of particular interest are the ***ground instances*** of an expression—those instances that have no variables. If e is an expression, then e_{gr} is the set of all of its ground instances, and if E is a set of expressions, then E_{gr} is the set of all ground instances of all expressions in E.

An algorithm needs a systematic way of naming each substitution it uses with a finite expression. Since we are only concerned with substitutions that have finite domains, this can be accomplished in the following straightforward way. Suppose that the domain of a substitution is $\{x_1, x_2, \ldots, x_n\}$ and that each x_i is mapped to some expression t_i. Then this substitution is denoted by $\{t_1/x_1,\ t_2/x_2,\ \ldots,\ t_n/x_n\}$. This naming scheme is also used in the text.

Substitutions, being functions, can be composed. If θ and σ are substitutions then their composition, $\theta\cdot\sigma$, is $\lambda e.\sigma(\theta(e))$. In other words, $\theta\cdot\sigma$ is such that $e(\theta\cdot\sigma)=(e\theta)\sigma$ for all expressions e.

The following lemmas concerning substitution are crucial for this work but are stated without proof because they are so widely known. (See, for instance, Robinson (1979) or Loveland (1978).)

Substitution Lemmas

(1) The identity function, henceforth denoted by ϵ, is a substitution. Furthermore, $\theta\cdot\epsilon=\epsilon\cdot\theta=\theta$, for any substitution θ.

(2) If σ and θ are substitutions, then so is $\sigma\cdot\theta$.

(3) The composition of substitutions is associative. That is, $(\theta_1\cdot\theta_2)\cdot\theta_3=\theta_1\cdot(\theta_2\cdot\theta_3)$, for all substitutions θ_1, θ_2, and θ_3.

Because of this last lemma, parentheses are not needed when writing compositions of substitutions.

θ is said to be a *renaming substitution* iff for any variables, x and y, $x\theta$ and $y\theta$ are variables and $x\theta=y\theta$ iff $x=y$. Expression e_1 is a *variant* of expression e_2 iff $e_2\theta=e_1$ for some renaming substitution θ.

Given expressions e and e' we will often want to know whether e_{gr} and e'_{gr} intersect. This is equivalent to determining whether $\{e,e'\}$ is *unifiable,* in the following sense.

Definition: Unifier

Let E be a set of expressions and θ be a substitution. If $E\theta$ is a singleton then θ is said to be a *unifier* of E or, alternatively, θ is said to *unify* E.

Often we will want to compute the set of all unifiers of E, but this may be infinite. Luckily, we need not be concerned with substitutions that can be obtained from others by composition—we only need a (hopefully finite) basis from which we can generate all substitutions in the set. This can be done by ordering the substitutions and forming the basis of a set of substitutions from certain representatives of its maximal elements.

Definition: More General

Substitution θ_1 is more general than substitution θ_2 (written $\theta_1 \geq \theta_2$) iff $\theta_1\cdot\sigma=\theta_2$ for some substitution σ.

Note that $\geq$ is reflexive and transitive but *not* anti-symmetric. Hence, despite its typographic appearance, $\geq$ is not a partial order.

Definition: Complete Set

Let Θ' and Θ be sets of substitutions. Then Θ' is a *complete set* of Θ iff:

(1) Θ' is correct; if $\theta' \in \Theta'$ and $\theta' \geq \theta$ then $\theta \in \Theta$.

(2) Θ' is complete; if $\theta \in \Theta$ then for some $\theta' \in \Theta', \theta' \geq \theta$.

Not every set of substitutions has a complete set. For instance, if $\theta = \{f(y)/x\}$ then $\Theta = \{\theta\}$ does not have a complete set. For if it did, the completeness condition insists that such a set contains a substitution $\sigma \geq \theta$. However, the correctness condition would then be violated because Θ fails to contain every substitution less general than σ—for instance, $\{f(a)/x\}$.

When a set of substitutions does have a complete set, it has many. However, they are not all created equal; some are smaller than others. Let us consider an example. The set of unifiers of $\{x,a\}$ has many complete sets, including $\{\{a/x\}\}$ and $\{\{a/x\},\{a/x,b/y\}\}$. The second set is larger than the first because it contains redundant information; notably, one of its substitutions is less general than the other. Accordingly, when representing a set of substitutions with a complete set it is economical to use a complete set that is *most general* in the following sense:

Definition: Most General Set of Substitutions

A set of substitutions is *most general* if it does not contain two distinct substitutions such that one is more general than the other.

I often will be concerned with most general complete sets of the unifiers of some set of expressions E and therefore call such a set an MGCU of E. The above example states that the set of all unifiers of $\{a,x\}$ has a complete set. In fact, the set of all unifiers of any given set of expressions has a complete set. This is a consequence of Robinson's (1965) celebrated Unification Theorem, which also states that the Unification Algorithm computes MGCU's of cardinality 1.

Unification Theorem

Let E be any finite set of expressions. Then E is unifiable iff the Unification Algorithm so indicates upon termination. Moreover, the substitution σ then available as output is such that $\{\sigma\}$ is an MGCU of E.

Often, we will be concerned with the behavior of a substitution on only a certain set of variables. This motivates the definition of the *restriction* of a substitution.

Definition: Restriction of a Substitution
If V is a set of variables and θ is a substitution, then θ *restricted to* V, or θ/V, is the substitution such that for every variable v:

(1) if $v \in V$ then $v\theta/V = v\theta$.
(2) if $v \notin V$ then $v\theta/V = v$.

We will have occasion to use the following lemmas, which follow immediately from the definition of "restriction."

Restriction Lemmas
Let θ be a substitution, V be a set of variables, and e be an expression. Then:

(1) $VARS(e) \subseteq V$ implies $e\theta = e\theta/V$.
(2) $\theta/V \geq \theta$.
(3) $(\theta_1 \cdot \theta_2)/V \geq \theta_1/V \cdot \theta_2$.

3.3. The Specification of Retrievability of Schematic Sequents

Recall that our goal is to specify a retriever that responds to a queried sentence schema by supplying a substitution for the schematic variables in the sentence schema. This idea is now formalized starting with the definitions of "schematic variable" and "sentence schema."

Schematic variables are meta-linguistic variables that range over the terms of the object language (QFPC, in this case). Sentence schemas are identical to sentences except that schematic variables may appear anywhere that terms may. In particular, every ground sentence is a sentence schema that happens not to contain any schematic variables. Schematic sequents can be constructed from schematic sentences just as ground sequents are constructed from ground sentences.

Schematic variables are distinct from the so-called *logical variables,* object-language variables used for quantification. To reflect this distinction, schematic variables always wear hats, e.g., $\hat{x}$, $\hat{y}$, and $\hat{z}$. The distinction between schematic and logical variables is not important in this chapter because QFPC sequents do not contain logical variables. However with the introduction of quantification in the next chapter, logical variables will occur in sequents and the distinction will be vital. To help keep the distinction clear, I use the expression $SVARS(e)$ to refer to the set of schematic variables in e and $LVARS(e)$ to refer to the set of logical variables in e.

Though sentence schemas are neither true nor false, the domain of the retrievability relation can be extended to include schematic sequents. The motivation for this extension is the idea that a schematic query should succeed iff one of its ground instances is retrievable. In addition, schematic sentences can be contained in a KB with the understanding that this is equivalent to a KB that contains all instances of its schemas. The specification of retrievability of schematic sequents is stated more precisely by the following definition.

Definition: Retrievability of Schematic Sequents
Let $kb \Rightarrow q$ be a schematic sequent of QFPC and θ be a ground substitution for $SVARS(q)$. Then q is retrievable from kb with answer θ iff $kb_{gr} \models_{RP} q\theta$. Additionally, θ is said to be an answer to $kb \Rightarrow q$.

Notice that a retriever meeting this specification is sound in that any answer θ to $kb \Rightarrow q$ is such that $kb_{gr} \models_T q\theta$. Also notice that if a ground query is retrievable, it is retrievable with only one answer, ϵ. What's more, this definition of retrievability and the definition of the last chapter coincide over the set of ground sequents. That is, a ground sequent $kb \Rightarrow q$ is in the above retrievability relation iff $kb \models_{RP} q$.

Given a sequent we could consider various problems: the problem of deciding whether it has an answer, the problem of finding some specified number of answers to it, or the problem of finding all answers to it. Since a solution to the last of these would provide a solution to the previous two, we henceforth only consider the problem of finding all answers to a sequent. Therefore, every retrieval problem can be characterized by a finite schematic sequent; $kb \Rightarrow q$ characterizes the problem of finding every θ such that q is retrievable from kb with answer θ.

In response to a query, the obvious thing for a retriever to do is to hand back a list of all answers. However, this is impossible because many schematic sequents have an infinite set of answers. For example, among the answers to $P(f(\hat{x})) \Rightarrow P(\hat{y})$ are

$$\{f(a)/\hat{y}\},\quad \{f(f(a))/\hat{y}\},\quad \{f(f(f(a)))/\hat{y}\},\quad \ldots \tag{3.11}$$

The solution to this difficulty lies in finding a way to finitely characterize infinite sets of answers. Fortunately such a method is at hand. Notice that the substitutions in (3.11) are all less general than $\sigma = \{f(\hat{z})/\hat{y}\}$. Moreover, every ground substitution for $\{\hat{y}\}$ that is less general than σ is an answer. Therefore, σ can be used to characterize an infinite set of answers, and accordingly is called a *generalized answer*. Note, however that σ itself is *not* an answer because $\hat{y}\sigma$ is not ground.

Definition: Generalized Answer
Let $kb \Rightarrow q$ be a schematic sequent of QFPC and γ be a substitution for $SVARS(q)$. Then γ is a *generalized answer* to $kb \Rightarrow q$ iff every ground substitution $\theta \leq \gamma$ for $SVARS(q)$ is an answer to $kb \Rightarrow q$.

The set of generalized answers to a sequent is no smaller than the set of answers to that sequent; in fact, every answer is a generalized answer. However, generalized answers have the important property that they can be characterized by finite complete sets. Furthermore, given a complete set of generalized answers Γ to $kb \Rightarrow q$, it is easy to recover the set of all answers: it is simply

$$\{\theta \mid \theta \text{ is a ground substitution for } SVARS(q), \text{ and } \theta \leq \gamma \text{ for some } \gamma \in \Gamma\}$$

This, then, provides the solution: I characterize the set of answers to a sequent by a finite complete set of its generalized answers. This characterization depends on *every finite schematic sequent of QFPC having a finite complete set of generalized answers.* Indeed, this is the case and is so proved in the next section where I present a retrieval algorithm and prove that, when input a finite schematic sequent of QFPC, the algorithm always halts and outputs a finite complete set of generalized answers to the input sequent.

3.4. The Retrieval Algorithm

This section addresses the problem of computing a complete set of generalized answers to a schematic sequent. Once again the solution is divided into two stages: the first transforms an arbitrary sequent into a CNF sequent and the second computes a complete set of generalized answers to a CNF sequent.

The Conjunctive Normal Transformation can be applied to sentence schemas in the same way that it is applied to ground sentences; a schematic variable is treated just like any other term. As the following theorem tells us, the CNT preserves retrievability and therefore every retrieval problem can be transformed into a CNF retrieval problem. One observation lies at the root of this theorem's proof: the composition of the CNT with any substitution is commutative. In other words, $CNT(\alpha\theta) = CNT(\alpha)\theta$, for every substitution θ and sentence schema α.

CNT Theorem for Schematic Sequents
Let $kb \Rightarrow q$ be a schematic sequent of QFPC. Then θ is an answer to $kb \Rightarrow q$ iff it is an answer to the CNT of $kb \Rightarrow q$.

Proof
Consider the two conditions necessary for q to be retrievable from kb with answer θ. First, θ must be a ground substitution for $SVARS(q)$, which obviously is the case iff it is a ground substitution for $SVARS(CNT(q))$. Second, kb_{gr} must RP–entail $q\theta$. By the Conjunctive Normal Transformation Theorem, this is the case iff $CNT(kb_{gr}) \models_{RP} CNT(q\theta)$, which, by commutativity, is the case iff $CNT(kb)_{gr} \models_{RP} CNT(q)\theta$. ∎

Once a schematic sequent has been transformed into CNF, the Schematic Sequent Retrieval Algorithm (or SSRA) below can be used to compute a complete set of general answers to it. The essence of the SSRA is that its computations are, in some sense, schemas for the computations performed by the Ground Sequent Retrieval Algorithm. This method of handling sentence schemas is commonplace in the field of automated deduction, where the schematic computations are said to *lift* the ground computations. The correctness of the SSRA is established via the Lifting Theorem, which says that every ground instance of a schematic proof is a ground proof and every ground proof is an instance of some schematic proof.

Schematic Sequent Retrieval Algorithm

Input: $kb \Rightarrow q$, a schematic CNF sequent of QFPC

Output: SUCCESS or FAILURE; if SUCCESS then a substitution is also output

```
(1)  let $s$ = number of conjuncts in $q$
(2)  let $q_i$ = $i^{th}$ conjunct of $q$ $(1 \leq i \leq s)$
(3)  let $K$ be the set containing every conjunct of every sentence in $kb$
(3') let $\Theta_0 = \epsilon$
(4)  for $i = 1$ to $s$ do
(5)      choose to do either step A) or step B)
(6)      A) choose $p_i$, a positive literal in $q_i$
(7)         choose $n_i$, the complement of a negative literal in $q_i$
(8)         if $n_i\Theta_{i-1}$ and $p_i\Theta_{i-1}$ are unifiable
(9)         then let $U_i$ be any MGCU of $n_i\Theta_{i-1}$ and $p_i\Theta_{i-1}$
                 choose $\theta_i \in U_i$
                 let $\Theta_i = (\epsilon \cdot \theta_1 \cdot \theta_2 \cdots \theta_i)/VARS(q)$
(10)        else FAIL
(11)     B) choose $b_i \in K$ and rename it so that $VARS(b_i) \cap VARS(q\Theta_{i-1}) = \emptyset$
(12)        let $l_{i,1}, \ldots, l_{i,m}$ be the literals of $b_i$
(13)        choose $F_i$, a total function from $LITERALS(b_i)$ to $LITERALS(q_i)$
(14)        if $\langle l_{i,1}, \ldots, l_{i,m} \rangle$ and $\langle F_i(l_{i,1}), \ldots, F_i(l_{i,m}) \rangle\Theta_{i-1}$ are unifiable
(15)        then let $U_i$ be any MGCU of $\langle l_{i,1}, \ldots, l_{i,m} \rangle$ and $\langle F_i(l_{i,1}), \ldots, F_i(l_{i,m}) \rangle\Theta_{i-1}$
                 choose $\theta_i \in U_i$
                 let $\Theta_i = (\epsilon \cdot \theta_1 \cdot \theta_2 \cdots \theta_i)/VARS(q)$
(16)        else FAIL
(17) SUCCEED and output $\Theta_s$
```

As before, the SSRA defines a provability relation.

Definition: SSRA–provable

Let $kb \Rightarrow q$ be a schematic CNF sequent of QFPC. Then, q is *SSRA-provable* from kb (written $kb \vdash_{SSRA} q$) with *extracted answer* θ iff there is some sequence of choices for which the Schematic Sequent Retrieval Algorithm halts with SUCCESS and outputs θ when input $kb \Rightarrow q$.

The principal result of this section is that every extracted answer is a generalized answer and that the set of extracted answers produced by all proofs is a complete set. Before turning our attention to the series of results that lead to this conclusion, let us examine the SSRA itself, particularly by comparison to the GSRA.

The Ground Sequent Retrieval Algorithm and the Schematic Sequent Retrieval Algorithm have the same form. This is reflected by the step numbering, which indicates the correspondence of the steps in the two algorithms.

The most prominent addition incorporated in the SSRA is the sequence of substitutions $\Theta_0, \Theta_1, \ldots, \Theta_s$. Each Θ_i is called the i^{th} *partial extracted answer* and, as previously defined, Θ_s is called the extracted answer. Each Θ_i is a maximally general substitution that can be applied to q in order that the computation can complete i iterations making the choices that it did. For $0 \leq i \leq s$, $\Theta_i = (\epsilon \cdot \theta_1 \cdots \theta_i)/VARS(q)$ where, as we shall see, θ_i is a maximally general substitution that can be applied to $q\Theta_{i-1}$ in order that the computation can complete the i^{th} iteration given the choices that have been made on the first $i-1$ iterations. Accordingly, Θ_0 is initialized to ϵ just before the start of the first iteration (in step (3$'$) and each successive Θ_i is computed during the i^{th} iteration. Note also that $\Theta_{i-1} \leq \Theta_i$, for $1 \leq i \leq s$.

Now let us consider how each θ_i, and consequently Θ_i, is derived. The equality tests of (8) and (14) have been replaced with unifiability tests. This is what one expects; two schematic expressions are unifiable iff some ground instance of the first schema is equal to a ground instance of the second. In each case, if the unifiability test succeeds, then θ_i is chosen from among the elements of an MGCU. θ_i, as computed in steps (9) and (15), is a maximally general substitution that allows the expressions of steps (8) and (14), respectively, to be unified. Once θ_i is computed, steps (9) and (15) each proceed to compute Θ_i.

The only other difference between the SSRA and the GSRA is located in step (11). Here, the schematic algorithm renames b_i in order to avoid variable name clashes.

The sequence of Θ_i's computed by a schematic computation encodes the relationship between the schematic computation and all of the ground computations that are instances of the schema. Consider the example of retrieving $q = P(\hat{w},\hat{z},\hat{u}) \wedge Q(\hat{w},\hat{z},\hat{u})$ from $kb = \{P(\hat{x},a,\hat{v}),\ Q(a,\hat{x},\hat{v})\}$. The first iteration computes $\theta_1 = \{a/\hat{z},\ \hat{w}/\hat{x},\ \hat{u}/\hat{v}\}$ and $\Theta_1 = \epsilon \cdot \theta_1 / V = \{a/\hat{z}\}$. This indicates that *every* ground query that is an instance of $q\Theta_1 = P(\hat{w},a,\hat{u}) \wedge Q(\hat{w},a,\hat{u})$ could also have completed this first iteration. The second iteration must find the instances of $Q(\hat{w},a,\hat{u})$ that succeed and in doing so computes $\theta_2 = \{a/\hat{w}, \hat{u}/\hat{v}\}$ and $\Theta_2 = \{a/\hat{w}, a/\hat{z}\}$. This indicates that *every* ground query that is an instance of $q\Theta_2 = P(a,a,\hat{u}) \wedge Q(a,a,\hat{u})$ also could have completed the two iterations. Hence, as desired, Θ_2 is a generalized answer to $kb \Rightarrow q$.

Let us now observe that every step of the SSRA is effectively computable. To see this, we need only consider the operations that are part of the SSRA but not the GSRA:

(1) deciding whether two expressions are unifiable,
(2) finding an MGCU of two unifiable expressions,
(3) applying a substitution to an expression,
(4) restricting the domain of a substitution, and
(5) composing two substitutions.

The Unification Theorem tells us that (1) and (2) are finitely computed by the Unification Algorithm. Since substitutions produced by that algorithm have finite domains, they can be represented as sets such as $\{a/\hat{x},\ b/\hat{y}\}$. Using this representation (3), (4) and (5) can each be computed straightforwardly, and furthermore, the substitutions resulting from (4) and (5) also have finite domains.

We also can observe that the SSRA always terminates when input a finite sequent. As discussed in the previous chapter, this claim is contingent on every non-deterministic choice being made from a finite set. Except for the choices made in steps (9) and (15), this observation has already been made in the course of examining the GSRA. Steps (9) and (15) pose no problem because the Unification Theorem tells us that these choices are made from finite sets—indeed, singleton sets.

It may seem odd that the algorithm is stated in such a way that a non-deterministic choice is made from a singleton set. The algorithm turns its back on the Unification Theorem, and so does the theoretical analysis that follows. Only in the above discussion of termination is the size of MGCU's considered, and even there only their finiteness matters. By ignoring the size of the MGCU's, both the algorithm and the analysis achieve greater generality, generality that is exploited in Chapter 5. That chapter introduces a variety of unification for which non-singleton MGCU's exist. Since the results of this chapter are independent of the size of the MGCU's, they can be carried over intact and used in Chapter 5.

Steps (8) and (14) do not say how to decide unifiability nor do steps (9) and (15) say how the MGCU's are to be computed. Though the Unification Theorem points to the Unification Algorithm as one method that could be used, neither the algorithm nor the analysis insist on this. For example, the proof of the algorithm's correctness is not contingent on whether the MGCU is or is not one that could be computed by the Unification Algorithm. Though I do not do so, this approach could be taken one step further, for the correctness of the algorithm in no way depends on whether the complete sets of unifiers are most general. The use of complete sets that are not most general would only introduce redundancy into the search spaces of the algorithm. In the extreme case, the use of infinite complete sets would yield infinite search spaces and render the guarantee of termination null and void.

The SSRA implicitly defines a search space in the same way that the GSRA does. The only differences are that the arcs of an SSRA search space are labeled with an additional equation to show the value of θ_i, and that SUCCESS nodes are labeled with the corresponding extracted answer. By the argument above, the SSRA implicitly defines a finite search space for every finite sequent. Figure 3.1 displays an example of an SSRA search space, that of $P(a),\ R(\hat{z}) \Rightarrow (P(\hat{x}) \vee Q(\hat{x})) \wedge (R(\hat{y}) \vee \neg R(\hat{x}))$. To expedite the display and the discussion that follows, the labels on search space arcs are condensed and written as four-tuples in one of two forms:

$\langle A, p_i, n_i, \theta_i \rangle$, or

$\langle B, b_i, \langle F_i(l_{i,1}), \ldots, F_i(l_{i,m}) \rangle, \theta_i \rangle$.

Examination of the SSRA and the GSRA shows that they behave identically when input a ground sequent. This is stated by the Ground Equivalence Theorem.

Ground Equivalence Theorem

Let $kb \Rightarrow q$ be a ground CNF sequent of QFPC. Then $kb \vdash_{\overline{GSRA}} q$ iff $kb \vdash_{\overline{SSRA}} q$. Moreover, the Ground Sequent Retrieval Algorithm and the Schematic Sequent Retrieval Algorithm implicitly define isomorphic search spaces. They differ only in that every arc in the schematic search space has an additional label of the form "$\theta_i = \epsilon$".

Now that we see how the GSRA relates to the SSRA on ground sequents, let us see how the SSRA on ground sequents relates to the SSRA on schematic sequents. I have already mentioned that computations with schematic sequents are themselves schematic for computations on ground sequents. This notion is captured by the definition of *lifting*.

Definition: Lifting

Let D be a length s derivation from $kb \Rightarrow q$ with partial extracted answers $\Theta_0, \Theta_1, \ldots, \Theta_s$ and let D' be a length s derivation from $kb' \Rightarrow q'$ with partial extracted answers $\Theta_0', \Theta_1', \ldots, \Theta_s'$. Then D *lifts* D' provided that these conditions are met for $1 \leq i \leq s$:

- The i^{th} arc of D is labeled with step A iff the i^{th} arc of D' is.
- If the i^{th} arc of D is labeled $\langle A, n_i, p_i, \theta_i \rangle$ and the i^{th} arc of D' is labeled $\langle A, n_i', p_i', \theta_i' \rangle$ then $\langle n_i', p_i' \rangle$ is an instance of $\langle n_i, p_i \rangle$.
- If the i^{th} arc of D is labeled $\langle B, b_i, \langle \phi_1, \ldots, \phi_m \rangle, \theta_i \rangle$ and the i^{th} arc of D' is labeled $\langle B, b_i', \langle \phi_1', \ldots, \phi_m' \rangle, \theta_i' \rangle$ then b_i' is an instance of b_i, and $\langle \phi_1', \ldots, \phi_m' \rangle$ is an instance of $\langle \phi_1, \ldots, \phi_m \rangle$.
- $q'(\epsilon \cdot \theta_1' \cdots \theta_i')$ is an instance of $q(\epsilon \cdot \theta_1 \cdots \theta_i)$, i.e., $\Theta_i' \leq \Theta_i$.

For example, the proof displayed along the left path of Figure 3.1 lifts the (one and only) proof of $P(a)$, $R(a)$, $R(b) \Rightarrow (P(a) \vee Q(a)) \wedge (R(a) \vee \neg R(a))$ and the one along the right path lifts the (one and only) proof of $P(a)$, $R(a)$, $R(b) \Rightarrow (P(a) \vee Q(a)) \wedge (R(b) \vee \neg R(a))$. Figures 3.2 and 3.3 show these proofs.

$\langle B,\ P(a),\ \langle P(\hat{x})\rangle,\ \{a/\hat{x}\}\rangle$	$\langle B,\ P(a),\ \langle P(a)\rangle,\ \epsilon\rangle$
$\langle A,\ R(\hat{x}),\ R(\hat{y}),\ \{a/\hat{y}\}\rangle$	$\langle A,\ R(a),\ R(a),\ \epsilon\rangle$
SUCCEED with $\{a/\hat{x},\ a/\hat{y}\}$	SUCCEED with ϵ

Left: Proof of $P(a),\ R(\hat{z}) \Rightarrow (P(\hat{x}) \vee Q(\hat{x})) \wedge (R(\hat{y}) \vee \neg R(\hat{x}))$
Right: Proof of $P(a),\ R(a),\ R(b) \Rightarrow (P(a) \vee Q(a)) \wedge (R(a) \vee \neg R(a))$

Figure 3.2: Proof at Left Lifts Proof at Right

$\langle B,\ P(a),\ \langle P(\hat{x})\rangle,\ \{a/\hat{x}\}\rangle$	$\langle B,\ P(a),\ \langle P(a)\rangle,\ \epsilon\rangle$
$\langle B,\ R(\hat{z}),\ \langle R(\hat{y})\rangle,\ \{\hat{y}/\hat{z}\}\rangle$	$\langle B,\ R(b),\ \langle R(b)\rangle,\ \epsilon\rangle$
SUCCEED with $\{a/\hat{x}\}$	SUCCEED with ϵ

Left: Proof of $P(a),\ R(\hat{z}) \Rightarrow (P(\hat{x}) \vee Q(\hat{x})) \wedge (R(\hat{y}) \vee \neg R(\hat{x}))$
Right: Proof of $P(a),\ R(a),\ R(b) \Rightarrow (P(a) \vee Q(a)) \wedge (R(b) \vee \neg R(a))$

Figure 3.3: Proof at Left Lifts Proof at Right

Though ground proofs such as those shown above are of particular interest, they are not the only proofs that can be lifted. For instance, Figure 3.4 displays a non-ground proof that is lifted by the right path of Figure 3.1. Note that the proof in Figure 3.4 lifts the right proof of Figure 3.3.

$\langle B,\ P(a),\ \langle P(\hat{x})\rangle,\ \{a/\hat{x}\}\rangle$
$\langle B,\ R(b),\ \langle R(b)\rangle,\ \epsilon\rangle$
SUCCEED with $\{a/\hat{x}\}$

Proof of $P(a),\ R(a),\ R(b) \Rightarrow (P(\hat{x}) \vee Q(\hat{x})) \wedge (R(b) \vee \neg R(\hat{x}))$

Figure 3.4: A Non-Ground Proof that is Lifted

As a result of the lifting relationship between proofs, the Lifting Theorem is obtained.

Lifting Theorem

Let $kb \Rightarrow q$ be a schematic CNF sequent of QFPC and σ be a ground substitution for $SVARS(q)$. Then $kb_{gr} \vdash_{\overline{SSRA}} q\sigma$ iff for some $\gamma \geq \sigma$, $kb \vdash_{\overline{SSRA}} q$ with extracted answer γ.

Proof

I refer to a proof of $kb_{gr} \Rightarrow q\sigma$ as P' and a proof of $kb \Rightarrow q$ as P. Furthermore, I refer to the values produced by P' by superscripting them with $'$ in order to distinguish them from the values produced by P, which are not superscripted. For instance, $q_0', \ldots, q_s'$ are produced by P' while $q_0, \ldots, q_s$ are produced by P. Also, throughout this proof let $V = VARS(q)$. First I make the general observation that for $0 \leq i \leq s$, $\Theta_i = (\epsilon \cdot \theta_1 \cdot \cdots \theta_i)/V = (\Theta_{i-1} \cdot \theta_i)/V$.

if clause: Let Prop2(j) be the proposition:

> If proof P traverses the loop j times producing a partial extracted answer Θ_j that is more general than σ, then there is a proof P' that traverses the loop j times.

Prop2(s), which is the if–clause of this theorem, is proved by induction. Prop2(0) trivially holds; $\Theta_0 = \epsilon$, which is more general than any σ. Assuming Prop2(i–1) I now prove Prop2(i), for any $1 \leq i \leq s$. By the induction hypothesis, $\Theta_{i-1} \geq \sigma$. On the i^{th} iteration of the loop, P executes either step (A) or step (B). In each case I show that P' can successfully execute the same step.

Step (A): If P successfully executed step (A) then θ_i unifies $n_i\Theta_{i-1}$ and $p_i\Theta_{i-1}$. Hence $\Theta_{i-1} \cdot \theta_i$ unifies n_i and p_i. Because $DOM(\Theta_{i-1} \cdot \theta_i) \subseteq V$, $\Theta_{i-1} \cdot \theta_i = (\Theta_{i-1} \cdot \theta_i)/V$, which in turn equals Θ_i. So, Θ_i unifies n_i and p_i. Now P' can also choose step (A) and can choose $n_i' = n_i\sigma$ and $p_i' = p_i\sigma$. Since Θ_i is more general than σ and unifies p_i and n_i, σ also unifies them and hence $p_i' = n_i'$. Therefore P' can successfully complete the i^{th} iteration.

Step (B): If P chooses step (B), b_i and F_i then $\langle l_{i,1}, \ldots, l_{i,m} \rangle$ and $\langle F_i(l_{i,1}), \ldots, F_i(l_{i,m}) \rangle \Theta_{i-1}$ are unifiable by θ_i. Since $\Theta_i \geq \sigma$, let ρ be such that $\Theta_i \cdot \rho = \sigma$. Now, P' can also choose step (B) and can choose b_i' and F_i' so that $b_i' = b_i\theta_i\rho$ (we will see shortly that this choice is indeed a ground formula) and $F_i'(l_{i,j}') = F_i(l_{i,j})\sigma$, for $1 \leq j \leq m$. However, for $1 \leq j \leq m$, $F_i(l_{i,j})\sigma$ equals $F_i(l_{i,j})\Theta_i\rho$, which in turn equals $F_i(l_{i,j})\Theta_{i-1}\theta_i\rho$ because $VARS(F_i(l_{i,j})) \subseteq V$. Hence, for $1 \leq j \leq m$, $l_{i,j}' = l_{i,j}\theta_i\rho$ equals $F_i'(l_{i,j}') = F_i(l_{i,j})\Theta_{i-1}\theta_i\rho$. Therefore, P' can successfully execute step (B).

only-if clause: Let Prop1(j) be the proposition:

> If proof P' traverses the loop j times then there is a proof P that traverses the loop j times producing a partial extracted answer Θ_j that is more general than σ.

I prove Prop1(s), which is the only-if clause of this theorem, by induction. Prop1(0) trivially holds for $\Theta_0 = \epsilon$. Assuming Prop(i–1) I now prove Prop(i). On the i^{th} iteration of the loop P' executes either step (A) or step (B). Consider each possibility:

Step (A): If P' chooses step (A) and n_i' and p_i' then $n_i' = p_i'$. P can then choose to do step (A) and can choose n_i and p_i so that $n_i\sigma = n_i'$ and $p_i\sigma = p_i'$. Since $\Theta_{i-1} \geq \sigma$ (by the inductive hypothesis), there exists a substitution λ such that

$$\Theta_{i-1}\cdot\lambda = \sigma. \tag{3.12}$$

Also because $\Theta_{i-1} \geq \sigma$, $n_i\Theta_{i-1}$ and $p_i\Theta_{i-1}$ are unifiable. Moreover, one element of any MGCU of $n_i\Theta_{i-1}$ and $p_i\Theta_{i-1}$ is more general than λ. Let P choose it as θ_i. Now, from (3.12) it follows that $\Theta_{i-1}\cdot\theta_i \geq \sigma$ and so $\Theta_i \geq \sigma$. Therefore P can execute step (A) generating a partial extracted answer $\Theta_i \geq \sigma$.

Step (B): If P' chooses step (B) and b_i' and F_i' then $l_{i,j}' = F_i'(l_{i,j}')$, for $1 \leq j \leq m$. Then P can choose step (B) and b_i and F_i so that $F_i(l_{i,j})\sigma = F_i'(l_{i,j}')$, for $1 \leq j \leq m$, and $b_i\rho = b_i'$ for some substitution ρ whose domain is $VARS(b_i)$. Then $l_{i,j}\rho = F_i(l_{i,j})\sigma$, for $1 \leq j \leq m$. Because $\Theta_{i-1} \geq \sigma$ (by the inductive hypothesis), there is a substitution λ such that

$$\Theta_{i-1}\cdot\lambda = \sigma \tag{3.13}$$

Also because $\Theta_{i-1} \geq \sigma$, $l_{i,j}\rho$ and $F_i(l_{i,j})\Theta_{i-1}$ are unifiable. Because $DOM(\rho) \cap VARS(F_i(l_{i,j})\Theta_{i-1}) = \emptyset$, $l_{i,j}$ and $F_i(l_{i,j})\Theta_{i-1}$ are also unifiable. Moreover, one element from any one of their MGCU's is more general than λ. Let P choose it as θ_i. Now, from (3.13) it follows that $\Theta_{i-1}\cdot\theta_i \geq \sigma$ and so $\Theta_i \geq \sigma$. Therefore, P can execute step (B) producing a partial extracted answer Θ_i that is more general than σ. ■

SSRA Correctness Theorem

Let $kb \Rightarrow q$ be a schematic CNF sequent of QFPC and $\Gamma = \{\gamma | kb \vdash_{\overline{SSRA}} q$ with extracted answer $\gamma\}$. Then Γ is a complete set of general answers to $kb \Rightarrow q$.

Proof

θ is an answer to $kb \Rightarrow q$

iff $kb_{gr} \models_{\overline{RP}} q\theta$	(Def. of Schematic Retrievability)
iff $kb_{gr} \vdash_{\overline{GSRA}} q\theta$	(GSRA Correctness Theorem)
iff $kb_{gr} \vdash_{\overline{SSRA}} q\theta$	(Ground Equivalence Theorem)
iff $kb \vdash_{\overline{SSRA}} q$ with some extracted answer $\gamma \geq \theta$	(Lifting Theorem)

Therefore, if γ is an extracted answer then every ground substitution $\theta \leq \gamma$ for $SVARS(q)$ is an answer. So γ is a generalized answer. Going the other way, if θ is an answer then some $\gamma \geq \theta$ is an extracted answer. Hence Γ is a complete set of generalized answers. This proof is encapsulated in Figure 3.5. ■

As a consequence of this theorem we can now see the validity of the previously unsubstantiated claim that every finite schematic sequent has a finite complete set of general answers. Such a set is, of course, Γ of the Correctness Theorem. That Γ is finite if $kb \Rightarrow q$ is finite is obvious when one recalls that every finite sequent has a finite search space.

The Correctness Theorem does not claim that Γ is a *most-general* complete set and indeed, in general, it is not most general. Obviously, Γ would not be most general if in steps

(9) and (15) the SSRA used complete sets of unifiers that were not most general. Less obviously, this can happen even though MGCU's are used. Figure 3.1 shows that one extracted answer to $P(a)$, $R(\hat{z}) \Rightarrow (P(\hat{x}) \vee Q(\hat{x})) \wedge (R(\hat{y}) \vee \neg R(\hat{x}))$, $\{a/\hat{x}\}$, is more general than the other, $\{a/\hat{x},\ a/\hat{y}\}$. In some sense this situation is purely coincidental; it is not caused by some defect in the algorithm such as a failure to schematize to the most-general level. The second arcs of the two proofs are unrelated, being computed by different methods; they yield two answers such that one just happens to be more general than the other.

Before closing this section, we return to the issue of selection functions. Like the GSRA, the correctness of the SSRA does not depend on the order in which the conjuncts of a query are worked on. This is true of the SSRA for the same reason that it is true of the GSRA: since the SSRA is correct for any arbitrary CNF query, it is correct for every query obtained by permuting the conjuncts of that arbitrary query.

However, unlike the GSRA, the SSRA does not have the stronger property of decomposability. The conjuncts of the query $P(\hat{x}) \vee Q(\hat{x})$ cannot be retrieved independently of each other. In retrieving $P(\hat{x})$ on the first iteration, the substitution Θ_1 is computed, and the second iteration has the task of retrieving $Q(\hat{x})\Theta_1$. Whereas a ground query has the property of being retrievable if each of its conjuncts is, a non-ground query does not have this property. In particular, conjuncts sharing variables are not decomposable. For example, both $P(\hat{x})$ and $Q(\hat{x})$ are retrievable from $P(a) \wedge Q(b)$ but $P(\hat{x}) \wedge Q(\hat{x})$ is not.

Though the choice of selection function does not affect the correctness of the SSRA, it can have a drastic effect on efficiency. This is illustrated by an example drawn from Chat-80 (Warren, 1981; Warren and Pereira, 1982), a computer program that accesses a simple KB in order to respond to English queries about world geography. Chat-80's KB contains atomic sentences describing the sort of each geographical entity it knows about, for instance

$$Country(US),\ Country(Canada),\ Country(Mexico),\ Country(Iceland),\ \ldots$$
$$Ocean(Atlantic),\ Ocean(Pacific),\ Ocean(Indian),\ \ldots$$

It also contains an atomic sentence for every border shared by two of these entities:

$$Borders(US,Canada),\ Borders(US,Atlantic),\ Borders(Iceland,Atlantic),\ \ldots$$

To find all countries bordering the U.S. either query (3.14) or (3.15) could be issued.

$$Country(\hat{x}) \wedge Borders(US,\hat{x}) \tag{3.14}$$
$$Borders(US,\hat{x}) \wedge Country(\hat{x}) \tag{3.15}$$

Either query does the job, but the second does it more efficiently. Since there are only 5 entities bordering the U.S. but approximately 150 countries, it is simpler to generate the 5 bordering entities and check whether they are countries than to generate the 150 countries and check whether they border the U.S. The search spaces for these two queries are displayed below in Figures 3.6 and 3.7. The arcs of these search spaces are labeled only with the value of θ_i

instead of an entire 4-tuple.

I use Chat-80 to illustrate the importance of the selection function because the program contains a query-planning mechanism that can automatically replace a query like (3.14) with a query like (3.15). To do this, the query planner estimates the number of answers that there are to each conjunct in the query and places that conjunct first. Then this is repeated for the remaining conjuncts, taking into account that variables may become instantiated as a result of the substitutions computed by answering the previous conjuncts. For example, on the basis of its estimates that $Country(\hat{x})$ has 150 answers and $Borders(US,\hat{x})$ has only 5, the query planner chooses (3.15) over (3.14).

Chat-80's query planner has other capabilities, of which I illustrate only one more. When confronted with the query

$$Country(\hat{x}) \wedge Borders(US,\hat{x}) \wedge Country(\hat{y}) \wedge Borders(France,\hat{y})$$

(which asks for all countries bordering the U.S. and all countries bordering France) Chat-80 can decompose it into two independent queries:

$$Country(\hat{x}) \wedge Borders(US,\hat{x})$$

$$Country(\hat{y}) \wedge Borders(France,\hat{y})$$

Furthermore, it realizes that because of the shared variables, neither of these two queries can be decomposed further.

Chat-80's query planner provides an excellent example of some techniques that can be used to build smaller search spaces. I say nothing more on this issue other than to suggest that similar techniques could be used to reduce the search required to answer the wh-queries of this chapter.

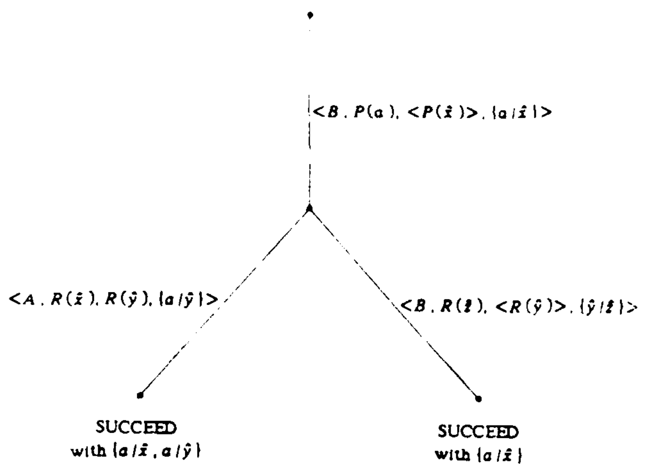


Figure 3.1: Search Space of $P(a), R(\hat{z}) \Rightarrow (P(\hat{x}) \vee Q(\hat{x})) \wedge (R(\hat{y}) \vee \neg R(\hat{x}))$

$$\begin{tikzcd}
kb \vdash_{\overline{SSRA}} q \text{ with extracted answer } \gamma \geq \theta \arrow[rr, leftrightarrow, dashed, "\text{SSRA Correctness Theorem}"] \arrow[d, leftrightarrow, "\text{Lifting Theorem}"'] & & \theta \text{ is an answer to } kb \Rightarrow q \arrow[d, leftrightarrow, "\text{Definition of Retrievability of Schematic Sequents}"] \\
kb_{gr} \vdash_{\overline{SSRA}} q\theta \arrow[r, leftrightarrow, "\text{Ground Equivalence Theorem}"'] & kb_{gr} \vdash_{\overline{GSRA}} q\theta \arrow[r, leftrightarrow, "\text{GSRA Correctness Theorem}"'] & kb_{gr} \models_{RP} q\theta
\end{tikzcd}$$

Figure 3.5: Proof of the SSRA Correctness Theorem

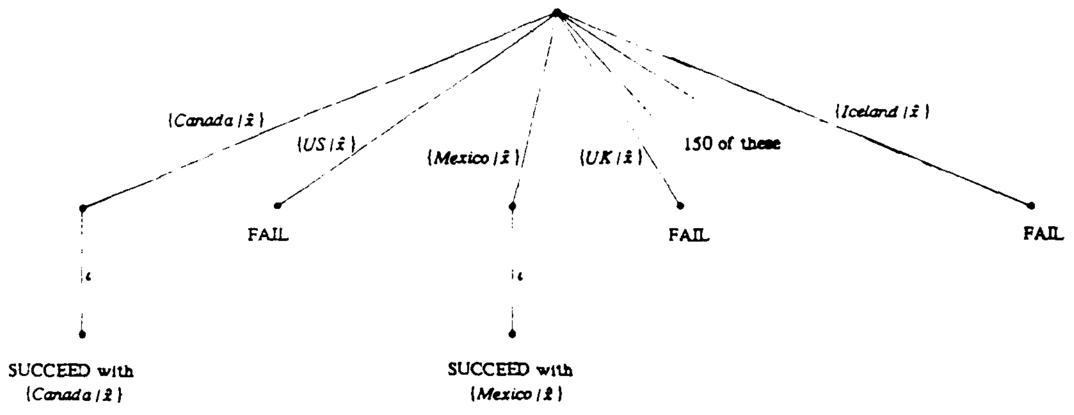


Figure 3.6: Search Space for the Query Country($\hat{x}$) ∧ Borders(US,$\hat{x}$)

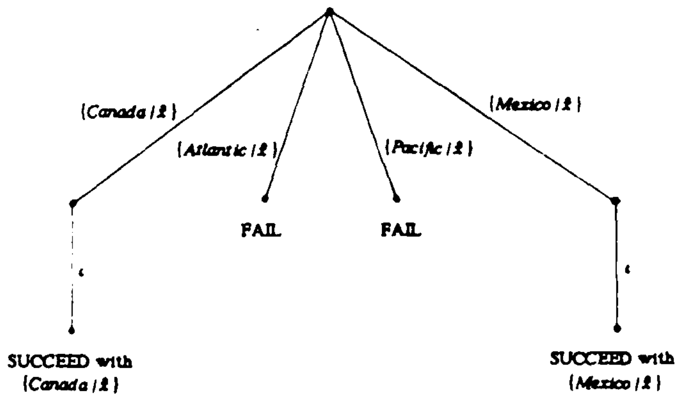


Figure 3.7: Search Space for the Query Borders(US, $\hat{x}$) ∧ Country($\hat{x}$)

Chapter 4

A Retriever for a Quantificational Language

This chapter extends the retriever of the previous chapter to deal with the entire first-order predicate calculus, quantifiers and all. The first section of this chapter shows that using RP to interpret quantifiers in the usual fashion yields an undecidable logic. Analysis of RP reveals how it allows chaining to slip in subtly, leading to undecidability. On the basis of this analysis, the second section develops a new logic, RQ, which is designed to agree with RP on retrievability in the absence of quantifiers, yet be decidable in their presence. Section 3 examines the properties of RQ, showing that it agrees with RP on propositional retrievability and that it has the important Strong Herbrand Property, which this chapter introduces. On the basis of these two properties, the fourth and final section demonstrates how the Schematic Sequent Retrieval Algorithm can be used to decide RQ-entailment.

The retriever that this chapter specifies operates on a language called the First-Order Predicate Calculus (FOPC), which is identical to QFPC except for the addition of quantifiers according to the following grammatical rule:

if ψ is a formula and x is a variable then $\exists x \psi$ and $\forall x \psi$ are formulas.

As is usual, FOPC sentences are closed FOPC formulas.

The logic developed here, RQ, is defined only for sentences in *prenex form*. Hence, from this point on, the word "sentence" only refers to prenex form sentences. A prenex form sentence contains no quantifiers in the constituent subformulas of any of its logical connectives. In terms of the surface syntax, quantifiers in a prenex form sentence appear at the left end of the sentence. Thus, $\forall x \exists y (P(x) \rightarrow R(x,y))$ is in prenex form whereas $\forall x (P(x) \rightarrow \exists y R(x,y))$ is not. A prenex form sentence is in *universal prenex form* iff it contains no existential quantifiers; likewise, it is in *existential prenex form* iff it contains no universal quantifiers.

A prenex form sentence divides neatly into two parts: a *prefix* containing all the quantifiers and a *matrix* containing the entire quantifier-free formula that appears to the right of the quantifiers. Hence, $\forall x \exists y (P(x) \rightarrow R(x,y))$ is composed of a prefix of $\forall x \exists y$ and a matrix of $(P(x) \rightarrow R(x,y))$. By convention, when I speak of instances of a prenex sentence I am actually referring to instances of its matrix.

4.1. An Inadequate Treatment of Quantification

This section investigates the outcome of giving quantifiers their *standard interpretation* in RP-models. By the "standard interpretation" for quantifiers I mean that a model M with domain D assigns a value to a quantified formula according to the following semantic equations, where $e[d/x]$ is a function identical to e with the possible exception that $e[d/x](x)=d$:

$$\text{True}\in[\![\forall x\alpha]\!]^{M,e} \text{ iff for all } d\in D,\ \text{True}\in[\![\alpha]\!]^{M,e[d/x]} \tag{4.1}$$
$$\text{False}\in[\![\forall x\alpha]\!]^{M,e} \text{ iff for some } d\in D,\ \text{False}\in[\![\alpha]\!]^{M,e[d/x]}$$

$$\text{True}\in[\![\exists x\alpha]\!]^{M,e} \text{ iff for some } d\in D,\ \text{True}\in[\![\alpha]\!]^{M,e[d/x]} \tag{4.2}$$
$$\text{False}\in[\![\exists x\alpha]\!]^{M,e} \text{ iff for all } d\in D,\ \text{False}\in[\![\alpha]\!]^{M,e[d/x]}$$

Since T uses the standard interpretation for quantifiers, every T-valuation is an RP-valuation. Thus, for FOPC as well as QFPC, $\models_{RP}$ is weaker than $\models_T$.

We can continue to view sentences as defining intensions, functions from A3S to A3, by considering a quantifier as a logical connective with an infinite number of arguments. It is easy to see that the intension of every sentence is monotonic, even in the presence of quantifiers. Hence, as before, the FOPC sentences valid in RP are precisely those valid in T.

This spells disaster because T-validity is only semi-decidable. Thus, it is impossible to determine whether an arbitrary FOPC sentence is retrievable from a KB even when the KB is empty. Furthermore, as the following theorem states, the *entire* $\models_T$ relation can be mapped into $\models_{RP}$.

T to RP Mapping Theorem[1]

Let $kb \Rightarrow q$ be a finite sequent of FOPC, let $P_1, \ldots, P_n$ be the predicates occurring in kb, and let Ψ_{kb} be the sentence

$$(\exists P_1(x_1,\ldots,x_{m_1}) \wedge \neg P_1(x_1,\ldots,x_{m_1})) \vee \cdots \vee (\exists P_n(x_1,\ldots,x_{m_n}) \wedge \neg P_n(x_1,\ldots,x_{m_n}))$$

Then $kb \models_T q$ iff $kb \models_{RP} q \vee \Psi_{kb}$.[2]

Proof

if clause: Assume $kb \models_{RP} q\vee\Psi_{kb}$. Then $kb \models_T q\vee\Psi_{kb}$, and because Ψ_{kb} is T-unsatisfiable, $kb \models_T q$.

only-if clause: Assuming $kb \models_T q$, I show that no RP-model is a countermodel to $kb \models_{RP} q\vee\Psi_{kb}$. An RP-model that does not satisfy kb is not a countermodel. If an RP-model that does satisfy kb is Tarskian then it satisfies q (by the assumption); otherwise it satisfies Ψ_{kb} (since it assigns {True, False} to some atomic formula). Therefore no RP-model is a countermodel to $kb \models_{RP} q\vee\Psi_{kb}$. ■

[1] This theorem and its proof were inspired by Patel-Schneider's (1985) closely-related First-Order Entailment Undecidability Theorem.

[2] That RP-validity and T-validity are identical is the special case of this theorem when $kb=\emptyset$

This result is surprising considering that RP does not sanction modus ponens and that all sentences are RP–satisfiable. What went wrong? Why hasn't the no–chaining restriction led to decidability in this quantificational logic? These questions can be answered by observing how this undecidability has arisen out of some fundamental properties of RP. The resulting insights are used to motivate the development of RQ.

To proceed with the analysis some of our tools need to be generalized. To begin with, the notion of entailment needs to be generalized to allow one set of sentences, A, to entail another set, B. As before, $A \models B$ means that no model satisfies A and falsifies B, and as before, a model satisfies a set of sentences iff it satisfies each member. Additionally, we now say that a model falsifies a set of sentences iff it falsifies each member.

Since entailment is now a relationship between sets of sentences, it is worth generalizing the definition of "sequent" to allow a set of sentences to appear in its consequent. These new sequents are called ***generalized sequents***, while mundane sequents with single–sentence consequents are called ***ordinary sequents***. As is consistent with the definition of entailment, I draw no distinction between a sequent whose consequent is a single sentence and one whose consequent is a singleton set. Hence, every ordinary sequent is also a generalized sequent. As with antecedents, set signs in a consequent are often omitted. Thus I often write "$P, Q \Rightarrow P, Q$" instead of "$\{P, Q\} \Rightarrow \{P, Q\}$", and "$P, Q \models P, Q$" instead of "$\{P, Q\} \models \{P, Q\}$".

Retrieval problems are always characterized by ordinary sequents. Non–ordinary sequents arise only in analyzing retrieval. The primacy of ordinary sequents over non–ordinary sequents is reflected in the choice of adopting the convention that only ordinary sequents are considered unless it is explicitly stated otherwise.

Though non–ordinary sequents do not arise directly in characterizing retrieval problems, they do arise in relating RP–entailment for FOPC sequents to that for QFPC sequents. This relation is established by the famous Herbrand Theorem. Though originally formulated for T, it holds equally well for many other logics, including RP. A logic for which the Herbrand Theorem holds is said to have the *Herbrand Property*. The Herbrand Theorem concerns sequents in a normal form known as Skolem Normal Form (or SNF).

Definition: Skolem Normal Form
A generalized sequent of FOPC is in *Skolem Normal Form* (SNF) iff all sentences in its antecedent are in universal prenex form and all sentences in its consequent are in existential prenex form.

Herbrand Theorem[3]

Let $kb \Rightarrow Q$ be a generalized SNF sequent of FOPC. Then, kb entails Q iff kb_{gr} entails Q_{gr}.

For T, both the Herbrand Theorem and its proof are widely published.[4] The proof, which is fairly general, is a product of the Herbrand Model Lemma and the use of the standard interpretation for quantifiers. Thus the proof for T can be used directly to prove the theorem for RP.

The Herbrand Theorem for RP relates RP–entailment for SNF sequents of FOPC to RP–entailment for sequents of QFPC. As such, the theorem provides a way of eliminating universal quantifiers from an antecedent and existentials from a consequent when considering questions of RP–entailment for SNF sequents. As an example of this theorem in action consider a language whose lexicon contains only the zero-place function symbols "a" and "b." The Herbrand Theorem reduces the question of whether (4.3) holds to the question of whether (4.4) holds.

$$\forall x P(x) \models_{RP} P(a) \vee P(b) \tag{4.3}$$

$$P(a), P(b) \models_{RP} P(a) \vee P(b) \tag{4.4}$$

The results of Chapter 2 bear on this latter implication. According to the RP–Decision Theorem for Facts, (4.4) holds, and, according to the GSRA Correctness Theorem, the Ground Sequent Retrieval Algorithm could be used to determine this.

In stark contrast to universal quantifiers in an antecedent, the occurrence of existentials in a consequent is extremely problematic. Observe that the Herbrand Theorem reduces the question of whether (4.5) holds to the question of whether 4.6 holds.

$$P(a) \vee P(b) \models_{RP} \exists x\, P(x) \tag{4.5}$$

$$P(a) \vee P(b) \models_{RP} P(a), P(b) \tag{4.6}$$

However, since (4.6) involves a generalized sequent, the results of Chapter 2, which only concern ordinary sequents, cannot be brought to bear. The GSRA can't be used to decide whether this implication holds. The No–Chaining Theorem doesn't apply.

Nonetheless, (4.6) does indeed hold. Any model that satisfies $P(a) \vee P(b)$ must satisfy either $P(a)$ or $P(b)$. In either case, the model does not falsify $\{P(a), P(b)\}$. This example illustrates that RP sanctions chaining among the elements of the consequent of a generalized sequent. Consequently, to respond to an existentially quantified query, a retriever specified by

[3] The usual statement of the Herbrand Theorem contains a claim about compactness. That aspect of the theorem is omitted from the present statement since it is not used in this work

[4] Once again Loveland (1978) and Robinson (1979) provide good expositions.

RP may have to chain together multiple instances of the query. Moreover, there is no bound on the number of instances that may need to be chained and therefore RP–validity is not decidable.

This example of chaining within a consequent is rendered benign by its simplicity, so let us consider a more–complex example where many instances of a query must be taken into account. Simply observe that the query

$$\exists x \, \neg I(0) \vee [I(x) \wedge \neg I(s(x))] \vee I(s(s(s(s(0))))) \tag{4.7}$$

is RP–valid. This may become more apparent by thinking of the interpretation in which "I" denotes a predicate that is true of precisely the integers, "s" denotes the successor function and "0" denotes zero. To form a valid (i.e., non–falsifiable) set, at least four ground instances of (4.7) are required: those generated by the substitutions $\{0/x\}$, $\{s(0)/x\}$, $\{s(s(0))/x\}$, and $\{s(s(s(0)))/x\}$. It is only by taking these four instances together that a valid set can be formed and it is because of this that chaining enters.

4.2. RQ: The Logic of a Retriever for a Quantificational Language

The RQ model theory for FOPC is developed with the goal of endowing the retrievability relation specified by RQ–entailment with two properties. Firstly, over the sentences of QFPC the retrievability relations specified by RQ and by RP should be identical. In this sense the retriever specified in this section should be an extension of the retriever specified in Chapter 2. Secondly, the retrievability relation specified by RQ should comply with the no–chaining restriction in all respects, including its treatment of the quantifiers.

The previous section points out that it is not sufficient merely to prohibit chaining among the facts of the KB, but that it is also necessary to prohibit it among instances of the query. In Chapter 2 the development of a logic that prohibits chaining among the facts of the KB was driven by an attempt to define a logic in which (4.8) does not hold.

$$P, \neg P \vee Q \models Q \tag{4.8}$$

In the same vein, this section's development of a logic that prohibits chaining among the instances of a query, is driven by an attempt to define a logic in which (4.9) does not hold.

$$P(a) \vee P(b) \models \exists x P(x) \tag{4.9}$$

A logic does not sanction the chaining of instances of a consequent if it has the *Strong Herbrand Property:*

> For any $kb \Rightarrow Q$, a generalized Skolem Normal Form sequent of FOPC, kb entails Q iff kb_{gr} entails a *single* element of Q_{gr}.

Whereas the Herbrand Property can reduce entailment with quantifiers to propositional entailment for generalized sequents, the Strong Herbrand Property can reduce it to propositional entailment for ordinary sequents. In a logic that has this property (4.9) holds only if

$$P(a) \vee P(b) \models P(a) \text{ or } P(a) \vee P(b) \models P(b) \tag{4.10}$$

(4.10) does not hold in T or any logic weaker than it.

The Strong Herbrand Property can be divided into a quantificational component and a propositional component. The quantificational component is the Herbrand Property. It reduces the problem of deciding whether (4.9) holds to the question of whether (4.11) does.

$$P(a) \vee P(b) \models \{P(a), P(b)\} \tag{4.11}$$

We can endow RQ with the Herbrand Property by giving quantifiers their standard interpretation. The propositional component of the Strong Herbrand Property is the *Minuteness Property*:

> A set of sentences entails a second set iff it entails a single element of the second set.

This property reduces the problem of deciding whether (4.11) holds to the problem of deciding whether (4.10) does.

Now consider how RQ can be defined so as to possess the Minuteness Property. If (4.11) does not hold then there must be a model that satisfies $P(a) \vee P(b)$ but falsifies both $P(a)$ and $P(b)$. That a disjunction can be satisfied when its disjuncts aren't is reminiscent of modal logic where $P(a) \vee P(b)$ may be necessarily true in spite of the non-necessity of $P(a)$ and of $P(b)$. (Symbolically, $\Box(P(a) \vee P(b)) \not\models \Box P(a), \Box P(b)$.) This insight motivates the possible-worlds style definition of RQ, to which we now turn.

Calling an RP-model a 3-setup,[5] an RQ-model is defined simply as a compatible set of 3-setups. 3-setups assign truth values to quantifier-free formulas in the same manner as RP models. The valuation associated with an RQ-model is defined in terms of the valuations associated with its 3-setups. Speaking very loosely, an RQ-model assigns a quantifier-free formula True if the formula is necessarily True, and False if it is possibly False. Quantifiers are interpreted in the standard way. (4.12), (4.13) and (4.14) define how prenex formulas are assigned truth values relative to a value assignment e and an RQ-model M whose common domain is D.

$$\begin{aligned} &\text{True} \in [\![\forall x \alpha]\!]^{M,e} \text{ iff for all } d \in D,\ \text{True} \in [\![\alpha]\!]^{M,e[d/x]} \\ &\text{False} \in [\![\forall x \alpha]\!]^{M,e} \text{ iff for some } d \in D,\ \text{False} \in [\![\alpha]\!]^{M,e[d/x]} \end{aligned} \tag{4.12}$$

$$\begin{aligned} &\text{True} \in [\![\exists x \alpha]\!]^{M,e} \text{ iff for some } d \in D,\ \text{True} \in [\![\alpha]\!]^{M,e[d/x]} \\ &\text{False} \in [\![\exists x \alpha]\!]^{M,e} \text{ iff for all } d \in D,\ \text{False} \in [\![\alpha]\!]^{M,e[d/x]} \end{aligned} \tag{4.13}$$

[5] This term derives from Belnap's use of the word "setup" to denote a 4-valued assignment.

and if α is quantifier–free

$$\text{True} \in [\![\alpha]\!]^{M,e} \text{ iff for every 3-setup } s \in M, \text{True} \in [\![\alpha]\!]^{s,e} \tag{4.14}$$
$$\text{False} \in [\![\alpha]\!]^{M,e} \text{ iff for some 3-setup } s \in M, \text{False} \in [\![\alpha]\!]^{s,e}$$

Regarding these semantic equations, several points are noteworthy. First of all (4.12) and (4.13) merely reiterate (4.1) and (4.2), and hence quantifiers receive their standard interpretation. Secondly, (4.12)-(4.14) do not assign values to non-prenex form sentences. Finally, to every prenex formula, a model and a value assignment assign a legitimate truth value, that is, a non-empty subset of {True, False}.

The valuation associated with an RQ model that contains only one 3-setup is identical to the valuation associated with that 3-setup. Hence every RP-valuation is an RQ-valuation and therefore RQ-entailment is weaker than RP-entailment.

Furthermore, RQ can generate valuations that RP does not. Among them are those valuations generated by countermodels to (4.9) and (4.11). Consider M, a model containing two 3-setups, s_1 and s_2, which have the common Herbrand domain $\{a,b\}$. Supposing that these 3-setups assign truth values to the atomic sentences as shown in (4.15), then they must also make the assignments shown in (4.16) and M must make the assignments shown in (4.17).

$$[\![P(a)]\!]^{s_1} = \{\text{True}\} \qquad [\![P(a)]\!]^{s_2} = \{\text{False}\} \tag{4.15}$$
$$[\![P(b)]\!]^{s_1} = \{\text{False}\} \qquad [\![P(b)]\!]^{s_2} = \{\text{True}\}$$

$$[\![P(a)\vee P(b)]\!]^{s_1} = \{\text{True}\} \qquad [\![P(a)\vee P(b)]\!]^{s_2} = \{\text{True}\} \tag{4.16}$$

$$[\![P(a)]\!]^{M} = \{\text{False}\} \tag{4.17}$$
$$[\![P(b)]\!]^{M} = \{\text{False}\}$$
$$[\![P(a)\vee P(b)]\!]^{M} = \{\text{True}\}$$
$$[\![\exists x\, Px]\!]^{M} = \{\text{False}\}$$

By satisfying $P(a)\vee P(b)$ and falsifying both $\exists x P(x)$ and $\{P(a),P(b)\}$, M demonstrates that, as desired, neither (4.9) nor (4.11) hold in RQ.

4.3. Properties of RQ

In a straightforward manner, this section proves that RQ does indeed have the properties that led to its design: that $\models_{RP}$ and $\models_{RQ}$ completely agree on the ordinary sequents of QFPC, and that in virtue of having the Minuteness Property and the Herbrand Property, RQ has the Strong Herbrand Property.

First consider the Herbrand Theorem for RQ. Since RQ-models are composed of RP models, the Herbrand Model Lemma holds in RQ as well as in RP. Because of this and RQ's use of the standard interpretation for quantifiers, the Herbrand Theorem also holds in RQ. Now consider the RP–RQ Equivalence Theorem and the Minuteness Theorem.

RP–RQ Equivalence Theorem
An ordinary sequent of QFPC is in $\models_{RP}$ iff it is in $\models_{RQ}$.

Proof
if clause: Trivial since every RP–model is an RQ–model.
only–if clause: Let $kb \Rightarrow b$ be an ordinary sequent of QFPC. Assuming that RQ–model M_{RQ} satisfies kb and falsifies q, I show that some 3–setup in M_{RQ} satisfies kb and falsifies q. Since M_{RQ} falsifies q, some 3–setup $s \in M_{RQ}$ falsifies q. Since M_{RQ} satisfies kb, so does every 3–setup in M_{RQ}. Therefore some 3–setup in M_{RQ} satisfies kb and falsifies q. ■

Minuteness Theorem (a.k.a. the Minuteness Theorem)
For any $kb \Rightarrow Q$, a generalized sequent of QFPC, $kb \models_{RQ} Q$ iff for some $q \in Q$, $kb \models_{RQ} q$.

Proof
if clause: Obvious.
only–if clause: I show that if $kb \not\models_{RQ} q$ for every $q \in Q$ then $kb \not\models_{RQ} Q$. I do this by assuming that for every $q \in Q$ there is a Herbrand model M_q that satisfies kb and falsifies q, and constructing a Herbrand model M^* that satisfies kb and falsifies Q. The Herbrand Lemma justifies my restricted attention to Herbrand models. Let M^* be the union of every M_q. Since they are Herbrand 3–setups, the 3–setups in M^* are compatible and therefore M^* is an RQ–model. Since each 3–setup $s \in M^*$ is in some M_q and M_q satisfies kb, s also satisfies kb. Hence M^* satisfies kb. Furthermore, every $q \in Q$ is falsified by some 3–setup in M_q, and hence by some 3–setup in M^*. Therefore M^* falsifies Q. ■

The Herbrand Theorem, the Minuteness Theorem and the RP–RQ Equivalence Theorem state the most fundamental properties of RQ. These results dovetail together to form composite results in a manner suggestive of the way that sentences combine to form paragraphs. The Herbrand Theorem relates $\models_{RQ}$ for FOPC sequents to $\models_{RQ}$ for generalized QFPC sequents, which the Minuteness Theorem relates to $\models_{RQ}$ for ordinary sequents, which the RP–RQ Equivalence Theorem relates to $\models_{RP}$ for ordinary sequents. RP–entailment for ordinary QFPC sequents is well-studied in Chapter 2. Results from that chapter, such as the No–Chaining Theorem, can be dovetailed onto the end of the above sequence of results.

Of these composite results, I now present two of the most interesting: the Strong Herbrand Theorem and the Generalized No Chaining Theorem.

Strong Herbrand Theorem

Let $kb \Rightarrow Q$ be a generalized SNF sequent of FOPC. Then $kb \models_{RQ} Q$ iff kb_{gr} RQ–entails some element of Q_{gr}.

Proof

Follows immediately from the Herbrand Theorem for RQ and the Minuteness Theorem. ■

Generalized No–Chaining Theorem

Let Φ be a set of QFPC sentences and A be a set of facts. Then $\Phi \models_{RQ} A$ iff for some $\phi \in \Phi$ and $\alpha \in A$, $\phi \models_{RQ} \alpha$.

Proof

$\Phi \models_{RQ} A$ iff for some $\alpha \in A$, $\Phi \models_{RQ} \alpha$ (Minuteness Theorem)
iff for some $\alpha \in A$, $\Phi \models_{RP} \alpha$ (RP–RQ Equivalence Theorem)
iff for some $\alpha \in A$ and $\phi \in \Phi$, $\phi \models_{RP} \alpha$ (No–Chaining Theorem for RP)
iff for some $\alpha \in A$ and $\phi \in \Phi$, $\phi \models_{RQ} \alpha$ (RP-RQ Equivalence Theorem) ■

4.4. Computing Retrievability

This section examines how the retrievability relation specified by RQ can be decided. As in previous chapters the use of a normal form divides the presentation in two. The first part presents the Skolem Normal Transformation, which transforms sequents into Skolem Normal Form. The second part demonstrates that by treating quantified variables schematically, the Schematic Sequent Retrieval Algorithm can be used to decide whether an SNF sequent is in $\models_{RQ}$.

The Skolem Normal Transformation (SNT) defined below maps an arbitrary sequent into an SNF sequent in such a way that, as stated by the Skolem Normal Transformation Theorem, its input is in $\models_{RQ}$ iff its output is. The SNT Theorem for T is well-published. Since its proof applies to RQ as well as to T, the theorem is stated without proof.

Definition: Skolem Normal Transformation

Let $kb \Rightarrow q$ be an FOPC sequent. Its Skolem Normal Transform is computed by the following steps.

(1) While kb contains a sentence ϕ that contains an existential quantifier do:

Observe that ϕ is of the form

$$\forall x_1 \forall x_2 \cdots \forall x_n \exists y\, \psi[y]$$

for some $n \geq 0$ and prenex-form formula $\psi[y]$. Choose ξ, some n-ary function symbol that does not occur in $kb \Rightarrow q$ and replace ϕ's occurrence in $kb \Rightarrow q$ with

$$\forall x_1 \forall x_2 \cdots \forall x_n\, \psi[\xi(x_1, \ldots, x_n)]$$

(2) While q contains a universal quantifier do:

Observe that q is of the form

$$\exists x_1 \exists x_2 \cdots \exists x_n \forall y\, \psi[y]$$

for some $n \geq 0$ and prenex-form formula $\psi[y]$. Choose ξ, some n-ary function symbol that does not occur in $kb \Rightarrow q$ and replace q's occurrence in $kb \Rightarrow q$ with

$$\exists x_1 \exists x_2 \cdots \exists x_n\, \psi[\xi(x_1, \ldots, x_n)]$$

Skolem Normal Transformation Theorem

A generalized sequent of FOPC is in $\models_{RQ}$ iff its Skolem Normal Transform is.

It should be noted that unlike the CNT, the SNT is not a transformation on sentences; it only applies to sequents as a whole. The SNT does not replace sentences with their equivalents; it merely preserves $\models_{RQ}$, which is all we are currently interested in.

Once a sequent is in SNF the Schematic Sequent Retrieval Algorithm can decide if it is in $\models_{RQ}$. Consider the SNF sequent of FOPC $kb \Rightarrow q$ and the schematic sequent $kb' \Rightarrow q'$ that is obtained by dropping its quantifiers and renaming its logical variables to schematic variables. From Chapter 3 we know that the SSRA can decide whether kb_{gr}' RP-entails some instance of q'. Therefore, because $kb_{gr} = kb_{gr}'$, $q_{gr} = q_{gr}'$, and RP and RQ are equivalent for ordinary sequents of QFPC, the SSRA also decides whether kb_{gr} RQ-entails some instance of q. Finally, according to the Strong Herbrand Theorem, deciding whether this last implication holds is equivalent to deciding whether $kb \models_{RQ} q$.

FOPC Retrieval Theorem

Let $kb \Rightarrow q$ be an ordinary SNF sequent of FOPC and let $kb' \Rightarrow q'$ be the schematic sequent that results from removing all quantifiers from $kb \Rightarrow q$. Then $kb \models_{RQ} q$ iff $kb' \models_{\overline{SSRA}} q'$.

Proof

$kb \models_{\overline{R}Q} q$ iff for some $qg \in q_{gr}$ $kb_{gr} \models_{\overline{R}Q} qg$	(Strong Herbrand Theorem)
iff for some $qg' \in q_{gr}'$ $kb_{gr}' \models_{\overline{R}Q} qg'$	(Since $kb_{gr} = kb_{gr}'$ and $q_{gr} = q_{gr}'$)
iff for some $qg' \in q_{gr}'$ $kb_{gr}' \models_{\overline{R}P} qg'$	(RP-RQ Equivalence Theorem)
iff q' is retrievable from kb'	(Def. of Schematic Retrievability)
iff $kb' \vdash_{\overline{SSRA}} q'$	(SSRA Correctness Theorem) ■

Chapter 5

A Retriever that Reasons about Taxonomies

This chapter develops a retriever that augments the no–chaining retriever of the previous chapter with the capability to chain in certain highly–constrained circumstances, namely when reasoning about taxonomies and when performing inheritance. This kind of inference has been performed by most semantic–network systems.

Let us consider an example that illustrates what I mean by "reasoning about taxonomies" and by "inheritance." Let *kb* be the knowledge base containing sentences (5.1)–(5.4).

$$Mustang(Olde\text{-}Black) \tag{5.1}$$

$$\forall x\ Mustang(x) \rightarrow Auto(x) \tag{5.2}$$

$$\forall x\ Auto(x) \rightarrow Vehicle(x) \tag{5.3}$$

$$\forall x\ Mustang(x) \rightarrow Built(Ford, x) \tag{5.4}$$

Among the sentences T–entailed by *kb* are

$$\forall x\ Mustang(x) \rightarrow Vehicle(x) \tag{5.5}$$

$$Auto(Olde\text{-}Black) \tag{5.6}$$

$$Built(Ford, Olde\text{-}Black) \tag{5.7}$$

The derivations of (5.5) and (5.6) from *kb* each exemplify what I have in mind when I speak of reasoning about a taxonomy while the derivation of (5.7) exemplifies inheritance.

The retriever of Chapter 4 performs none of these inferences; *kb* RQ–entails neither (5.5), (5.6), nor (5.7). If this is not clear then observe that any model M that has a domain of three elements—denoted by "*Olde-Black*," "*Ford*" and "*Touring-Machine*"—and that makes the following truth assignments satisfies each of (5.1)–(5.4) but falsifies each of (5.5)–(5.7).

$$[\![Mustang(Olde\text{-}Black)]\!]^M = \{\text{True}, \text{False}\}$$

$$[\![Mustang(Touring\text{-}Machine)]\!]^M = \{\text{True}\}$$

$$[\![Mustang(Ford)]\!]^M = \{\text{False}\}$$

$$[\![Auto(Olde\text{-}Black)]\!]^M = \{\text{False}\}$$

$$[\![Auto(Touring\text{-}Machine)]\!]^M = \{\text{True}, \text{False}\}$$

$$[\![Auto(Ford)]\!]^M = \{\text{False}\}$$

$$[\![Vehicle(Touring\text{-}Machine)]\!]^M = \{\text{False}\}$$

$$[\![Vehicle(Ford)]\!]^M = \{\text{False}\}$$

$$[\![Built(Ford, Olde\text{-}Black)]\!]^M = \{\text{False}\}$$

$$[\![Built(Ford, Touring\text{-}Machine)]\!]^M = \{\text{False}\}$$

The goal of this chapter is to extend RQ so that it can reason about taxonomies and perform inheritance without performing any other kind of chaining. A difficulty arises because certain desirable inferences—such as those exemplified above—are instances of the application of modus ponens, an inference rule that the retriever should not use indiscriminately. How then can a specification of retrieval distinguish the desirable forms of chaining from the undesirable forms?

The way out of this predicament is to follow a strategy often employed in the construction of semantic-network systems. In such systems specialized notation is used to encode certain kinds of information and specialized inference mechanisms are then used to deal with that notation. Typically, special nodes and links are used to encode taxonomic information. Accordingly, the retriever specified in this chapter operates on a language, called the Sorted First-Order Predicate Calculus (SFOPC), that extends FOPC with special notation for representing categories and for expressing that members of a category have particular properties.

5.1. The Sorted First-Order Predicate Calculus

We now turn to the definition of the Sorted First-Order Predicate Calculus. Since SFOPC has some of the features of a traditional sorted logic, I henceforth use the terminology of that field rather than more general terms such as "taxonomy." Roughly speaking a *sort* is a taxonomic category and a *sort symbol* denotes a sort.

In addition to the usual function and predicate symbols, the SFOPC lexicon contains a countable set of sort symbols. Typographically, sort symbols are written entirely in uppercase. Semantically, a sort symbol, like a monadic predicate, denotes a subset of the domain.

In addition to the ordinary kind of variables, SFOPC has *restricted variables*. A restricted variable is a pair, $x{:}\tau$, where x is a variable name and τ, often referred to as a *restriction*, is a finite set of sort symbols. Henceforth, the term "variable" refers generally to either an ordinary variable or a restricted variable.

A variable whose restriction is $\{S_1, S_2, \ldots, S_n\}$ is written as $x{:}S_1, S_2, \ldots, S_n$. The order of the sort symbols is irrelevant, and thus $x{:}S_1, S_2$ and $x{:}S_2, S_1$ are one and the same variable. For clarity, variables are often written in angle brackets, such as $\langle x{:}S_1, S_2, S_3\rangle$.

To avoid confusion I never write a formula containing two distinct variables that have the same variable name. That is, no formula contains variables $\langle x{:}\tau\rangle$ and $\langle x{:}\tau'\rangle$ where τ and τ' are distinct. This enables use of the following shorthand. If a formula has multiple occurrences of the same variable then the restrictions often are written on only the first occurrence. For example, (5.8) can be abbreviated as (5.9).

$$\forall x{:}S\ P(x{:}S) \lor Q(x{:}S) \tag{5.8}$$

$$\forall x{:}S\ P(x) \lor Q(x) \tag{5.9}$$

τ and ω are meta-linguistic symbols that always stand for restrictions. $\langle x{:}\tau_1,\tau_2, \ldots, \tau_n\rangle$ is a variable whose restriction contains precisely the sorts $\tau_1 \bigcup \tau_2 \bigcup \cdots \bigcup \tau_n$.

Put crudely, a Tarskian model for SFOPC is a Tarskian FOPC model to which an assignment to sort symbols has been grafted.[1] More precisely, a T-model for SFOPC is a pair $\langle M, A^S\rangle$ where M is a Tarskian FOPC model and A^S is a ***sort assignment***, a function that maps each sort symbol to a subset of the domain of M.

A sort symbol denotes the set of individuals that A^S assigns to it. A restriction also denotes a sort, the intersection of the sorts denoted by the sort symbols in the restriction. Therefore, if $M = \langle M', A^S\rangle$ is a T-model for SFOPC, and S is a sort symbol, and $\tau = \{S_1, \ldots, S_n\}$ is a restriction, then:

$$[\![S]\!]^{M,e} = A^S(S)$$

$$[\![\tau]\!]^{M,e} = [\![S_1]\!]^{M,e} \cap \cdots \cap [\![S_n]\!]^{M,e}$$

A restricted variable only ranges over the subset of the domain denoted by its restriction. Formally this is captured by the following semantic rules for quantifiers with restricted variables:

$$\begin{aligned} &\text{True} \in [\![\forall x{:}\tau\ \phi]\!]^{M,e} \text{ iff for every } d \in [\![\tau]\!]^{M,e}, \text{True} \in [\![\phi]\!]^{M,e[d/x]} \\ &\text{False} \in [\![\forall x{:}\tau\ \phi]\!]^{M,e} \text{ iff for some } d \in [\![\tau]\!]^{M,e}, \text{False} \in [\![\phi]\!]^{M,e[d/x]} \end{aligned} \tag{5.10}$$

$$\begin{aligned} &\text{True} \in [\![\exists x{:}\tau\ \phi]\!]^{M,e} \text{ iff for some } d \in [\![\tau]\!]^{M,e}, \text{True} \in [\![\phi]\!]^{M,e[d/x]} \\ &\text{False} \in [\![\exists x{:}\tau\ \phi]\!]^{M,e} \text{ iff for every } d \in [\![\tau]\!]^{M,e}, \text{False} \in [\![\phi]\!]^{M,e[d/x]} \end{aligned} \tag{5.11}$$

Notice that if S is a sort symbol that denotes the entire domain in some model M, then $[\![\forall\langle x{:}S\rangle\,\phi]\!]^{M,e} = [\![\forall x\,\phi]\!]^{M,e}$ and $[\![\exists\langle x{:}S\rangle\,\phi]\!]^{M,e} = [\![\exists x\,\phi]\!]^{M,e}$. Consequently, unsorted variables are often treated as sorted variables implicitly restricted to the "universal" sort. Also notice that if S denotes the empty set in some model, then that model assigns $\{\text{True}\}$ to $\forall\langle x{:}S\rangle\,\phi$ and $\{\text{False}\}$ to $\exists\langle x{:}S\rangle\,\phi$.

Now that quantifiers can be restricted to range over subsets of the domain we need a way to express relationships among these subsets. To do this, SFOPC is endowed with a special set of formulas, which are called *S-formulas* to distinguish them from the previous formulas, which are called *A-formulas*. S-formulas are constructed like ordinary formulas of FOPC except that they contain no ordinary predicate symbols; in their place are sort symbols acting as monadic predicate symbols. Hence, every atomic S-formula is of the form $S(t)$, where S is a sort symbol and t is an ordinary term. In the obvious way I use the terms *S-sentence* and *S-literal*.

S–formulas are assigned truth values as one would expect: an atomic formula $S(t)$ is assigned True if the domain element denoted by t is a member of the set denoted by S, and a molecular S–formula is assigned a value in the usual Tarskian manner.

SFOPC is no more expressive than FOPC; each sentence of SFOPC is T–equivalent to one (of about the same length) of FOPC. Clearly the addition of sort symbols does not make the language more expressive since they behave semantically like monadic predicate symbols. Nor does the addition of restricted variables enhance the expressiveness of the language. To see this, observe that if τ is the restriction $\{S_1, S_2, \ldots, S_n\}$ and e is a variable assignment that maps x to d, then

$$d \in [\![\tau]\!]^{M,e} \text{ iff } [\![S_1(x) \wedge S_2(x) \wedge \cdots \wedge S_n(x)]\!]^{M,e} = \text{True}$$

Because of this relationship I henceforth abbreviate the formula $S_1(t) \wedge S_2(t) \wedge \cdots \wedge S_n(t)$ as $\tau(t)$. Finally, observe that any formula containing restricted quantifiers can be rewritten to a T–equivalent one without restricted quantifiers on the basis of these equivalences:

$$\forall x{:}\tau\ \psi[x{:}\tau] \equiv_T \forall x\ \tau(x) \rightarrow \psi[x]$$

$$\exists x{:}\tau\ \psi[x{:}\tau] \equiv_T \exists x\ \tau(x) \wedge \psi[x]$$

The formula that results from removing all restricted quantifiers from a formula ϕ by this rewriting process is called the ***normalization*** of ϕ and is denoted by ϕ^N. If Φ is a set of formulas, then $\Phi^N = \{\phi^N \mid \phi \in \Phi\}$.

A KB now consists of a set of SFOPC sentences and a query always specifies an SFOPC sentence to be retrieved. It is often convenient to consider a KB as consisting of two components, an AKB containing all the A–sentences in the KB, and an SKB containing all the S–sentences in the KB. Sequents of SFOPC are often written in a form that exhibits the distinction between S–sentences and A–sentences. Namely, in an SFOPC sequent written in the form $\Sigma, akb \Rightarrow q$, the antecedent is divided into a set of S–sentences, Σ, and a set of A–sentences, akb. Similarly, entailments are written in the form $\Sigma, akb \models q$.

As before, only A–sentences in prenex form are considered and therefore "A–sentence" only refers to prenex form A–sentences. As this chapter progresses various restrictions are placed on the SKB.

It is worth noting that the SKB is a ***theory*** of a taxonomy, not a taxonomy.

5.2. RT: The Logic of a Taxonomic Retriever

This section defines RT, a model theory for SFOPC whose entailment relation serves as a retrievability relation. RT extends RQ to handle the syntactic extensions introduced in the last section. Recall that the resulting retrievability relation is to respect the no chaining restriction, except that it is to reason completely with taxonomic information. Thus the retriever should perform all taxonomic inferences sanctioned by the Tarskian semantics. We will see shortly that this is a simple notion to capture in a model theory.

Since the retriever is to reason fully about sorts, the interpretation of sort symbols—unlike the interpretation of predicate symbols—should not be weakened. So a sort symbol should be given its full Tarskian meaning. The definitions that follow are made with this in mind.

Just as a Tarskian SFOPC model is formed from a Tarskian FOPC model by appending a sort assignment to it, so an RT-model is formed from an RQ model. Thus, an RT-model is merely a pair consisting of an RQ-model and a sort assignment. An alternate view is that RT relaxes the Tarskian model theory for SFOPC in the same way that RQ relaxes the Tarskian model theory for FOPC.

An RT-model, $M = \langle M', A^S \rangle$, assigns the same semantic values to sort symbols and S-formulas as does any T-model whose domain is the common domain of M' and whose sort assignment is A^S. To prenex-form A-formulas containing no restricted quantifiers, M assigns assigns truth values in the same manner as M'. Hence, equations (4.12)-(4.14) describe how RT assigns truth values to such A-formulas. Restricted quantifiers are treated in RT as in T—that is, according to (5.10) and (5.11). Thus, ignoring unrestricted quantifiers, the semantic equations for RT are:

$$\begin{aligned}&\text{True} \in [\![\forall x{:}\tau\ \phi]\!]^{M,e} \text{ iff for every } d \in [\![\tau]\!]^{M,e}, \text{True} \in [\![\phi]\!]^{M,e[d/x]} \\ &\text{False} \in [\![\forall x{:}\tau\ \phi]\!]^{M,e} \text{ iff for some } d \in [\![\tau]\!]^{M,e}, \text{False} \in [\![\phi]\!]^{M,e[d/x]}\end{aligned} \tag{5.12}$$

$$\begin{aligned}&\text{True} \in [\![\exists x{:}\tau\ \phi]\!]^{M,e} \text{ iff for some } d \in [\![\tau]\!]^{M,e}, \text{True} \in [\![\phi]\!]^{M,e[d/x]} \\ &\text{False} \in [\![\exists x{:}\tau\ \phi]\!]^{M,e} \text{ iff for every } d \in [\![\tau]\!]^{M,e}, \text{False} \in [\![\phi]\!]^{M,e[d/x]}\end{aligned} \tag{5.13}$$

and if α is quantifier-free

$$\begin{aligned}&\text{True} \in [\![\alpha]\!]^{M,e} \text{ iff for every 3-setup } s \in M', \text{True} \in [\![\alpha]\!]^{s,e} \\ &\text{False} \in [\![\alpha]\!]^{M,e} \text{ iff for some 3-setup } s \in M', \text{False} \in [\![\alpha]\!]^{s,e}\end{aligned} \tag{5.14}$$

Notice that, as in RQ, these equations do not assign values to non-prenex form A-sentences.

In contrast to T, RT treats sort symbols and monadic predicate symbols differently. Whereas each monadic predicate is mapped to a function from the domain to the set of three truth values, each sort symbol is mapped to a subset of the domain—or, equivalently, to a function from the domain to {True, False}. Thus, S-formulas operate in a two-valued logic, A-formulas without restricted quantifiers operate in a three-valued logic, and A-formulas with restricted quantifiers operate in both. The logic sanctions chaining in those parts of the language that operate in two values but not in those parts that operate in three values.

In order to observe some chaining that RT sanctions, let us return to the simple taxonomic inferences considered at the beginning of this chapter. First observe that RT agrees with

RQ that none of (5.5)-(5.7) is entailed by kb, the KB containing sentences (5.1)-(5.4). However, if all of the sentences are written using $MUSTANG$, $AUTO$, and $VEHICLE$ as sort symbols, then the entailments do obtain. (5.1)-(5.4) can be written (T-equivalently) in SFOPC as three S-sentences and one A-sentence:

$$MUSTANG(Olde\text{-}Black) \tag{5.1$'$}$$

$$\forall x\ MUSTANG(x) \rightarrow AUTO(x) \tag{5.2$'$}$$

$$\forall x\ AUTO(x) \rightarrow VEHICLE(x) \tag{5.3$'$}$$

$$\forall x{:}MUSTANG\ Built(Ford, x) \tag{5.4$'$}$$

Furthermore (5.5) and (5.6) can be rewritten (T-equivalently) in SFOPC as two S-sentences:

$$\forall x\ MUSTANG(x) \rightarrow VEHICLE(x) \tag{5.5$'$}$$

$$AUTO(Olde\text{-}Black) \tag{5.6$'$}$$

And now, (5.1′)-(5.4′) RT-entail (5.5′), (5.6′), and (5.7). In particular, (5.2′) and (5.3′) together RT-entail (5.5′), and (5.1′) and (5.2′) together RT-entail (5.6′); both of these entailments operate entirely within two-valued Tarskian logic and obtain for the usual reasons. Additionally, (5.1′) and (5.4′) together RT-entail (5.7), but not as obviously. Consider any variable assignment e, and any model M that satisfies both (5.1′) and (5.4′). Let OB be $[\![Olde\text{-}Black]\!]^{M,e}$. Since M satisfies (5.1′), $OB \in [\![Mustang]\!]^{M,e}$. M also satisfies (5.4′), and thus by equation (5.12), $\text{True} \in [\![Built(Ford,x)]\!]^{M,e[OB/x]}$.[1] Consequently, $\text{True} \in [\![Built(Ford,Olde\text{-}Black)]\!]^{M,e}$. That is, M satisfies "$Built(Ford,Olde\text{-}Black)$."

5.3. An Approach to Computing with Restricted Quantifiers

The primary goal of the remainder of this chapter is to develop an algorithm that implements the retriever specified in the previous section—that is, a procedure for deciding whether any given ordinary finite SFOPC sequent is in the RT-entailment relation. However, before proceeding with the details of the technical developments it is worth pausing to overview the structure of these developments. In order to present a clear picture, this overview omits certain secondary, though important, points.

The approach used to develop a retrieval algorithm that operates on a language with restricted quantifiers and a sort theory mimics the approach used in Chapters 3 and 4 to develop a retrieval algorithm that operates on a language with ordinary quantifiers. So, let us begin by recapping the approach to ordinary quantifiers.

[1] It may or may not be the case that $\text{False} \in [\![Built(Ford,Olde\text{-}Black)]\!]^{M,e}$

First, the notion of substitution was developed and then used to form the link between schematic sentences and their ground instances. Then the Ground Sequent Retrieval Algorithm was "lifted" to form the Schematic Sequent Retrieval Algorithm; this lifting operation primarily involved replacing each test for equality between expressions with a test for unifiability, which, if successful, yields a complete set of unifiers. The Lifting Theorem proved that the SSRA does indeed treat schematic sentences as sentence schemas, that is, as representatives for their ground instances. Thus, it was established that the SSRA could be used to handle quantified sentences by removing their quantifiers and replacing their quantified variables with schematic variables.

The remainder of this chapter proceeds along the same lines, except that instead of working with ordinary variables it works with restricted variables, both quantified and schematic. First, Section 5.4 introduces the notion of a substitution being *well sorted*. Informally, a substitution is well sorted relative to a sort theory if it maps each variable to a term that satisfies the restriction associated with the variable. Well-sortedness must be considered relative to a sort theory because it is the sort theory that determines which terms satisfy which restrictions.

Building upon the notion of well sortedness, Section 5.5 examines the properties of RT. The most important result of that section, the Sorted Herbrand Theorem, relates the retrievability of $\Sigma, akb \Rightarrow q$, under certain circumstances, to the retrievability of $akb_{\Sigma gr} \Rightarrow q_{\Sigma gr}$, where $akb_{\Sigma gr}$ and $q_{\Sigma gr}$ are the ground instances of akb and q that are obtained by substitutions that are well sorted relative to Σ. Hence, sentences with restricted quantifiers can be treated as sorted schematic sentences, schematic sentences in which the schematic variables have restrictions associated with them.

Section 5.6 then takes up the task of developing a retriever for sorted schematic sequents. The algorithm itself, the Sorted Schematic Sequent Retrieval Algorithm (SSSRA), is identical to the SSRA with the exception that it uses well sorted unifiers wherever the SSRA uses arbitrary unifiers. The Sorted Lifting Theorem states that the SSSRA deals with sorted schematic sentences as it would deal with the set of all well sorted instances of the schema. This theorem is proved by systematically modifying the proof of the Lifting Theorem of Chapter 3. With this in place, it is easy to see that retrievability of sequents with restricted quantifiers can be decided by removing all quantifiers, replacing restricted quantified variables with restricted schematic variables, and handing the resulting sorted schematic sentence to the SSSRA. Justification for doing this is provided by the Sorted Lifting Theorem and the Sorted Herbrand Theorem.

We now consider well sorted substitutions and unifiers, which form the foundation of this approach to restricted quantifiers.

5.4. Well Sorted Substitutions and Unifiers

According to the previously-stated informal definition, a substitution is well sorted relative to a sort theory if it maps each variable to a term that satisfies the restriction associated with the variable. More precisely, a substitution θ is well sorted relative to a sort theory Σ if, and only if, for every variable $x{:}\tau$, $\langle x{:}\tau\rangle\theta$ is a term t such that $\Sigma \models \forall(\tau(t))$.

Two special cases of this definition are worth noting. If θ is well sorted relative to Σ and maps $x{:}\tau$ to a ground term t, then it must be that $\Sigma \models \tau(t)$. In other words, Σ must entail that t is of sort τ. If θ maps $x{:}\tau$ to a variable $y{:}\omega$ then it must be that $\Sigma \models \forall y{:}\omega\ \tau(y)$. That is, Σ must entail that ω is a subset of τ.

Expression e' is said to be a well sorted instance of e relative to Σ if $e' = e\theta$, for some substitution θ that is well sorted relative to Σ. In the obvious way, I speak of well sorted ground instances of a formula and write $e_{\Sigma gr}$ to denote the set of all ground instances of e that are well sorted relative to Σ.

Since they are substitutions, well sorted substitutions enjoy all the properties possessed by substitutions in general. So, for example, Substitution Lemma (3) of Chapter 3, which says that composition of substitutions is associative, trivially holds for well sorted substitutions. For other reasons, the correlates of Substitution Lemmas (1) and (2) hold for well sorted substitutions.

Well Sorted Substitution Lemmas

(1) The identity substitution, ϵ, is a well sorted relative to any sort theory.

(2) If σ and θ are well sorted substitutions relative to Σ, then so is $\sigma\cdot\theta$.

Proof

(1): ϵ maps every variable $x{:}\tau$ to itself and any Σ entails $\forall x{:}\tau\ \tau(x)$ since $\forall x{:}\tau\ \tau(x)$ is a valid sentence.

(2): I assume θ and σ are Σ-well sorted, and show that for any variable, $\langle x{:}$ $\Sigma \models \forall\tau(\langle x{:}\tau\rangle\theta\sigma)$. Let $\phi[\langle y_1{:}\tau_1\rangle, \ldots, \langle y_n{:}\tau_n\rangle]$ be $\langle x{:}\tau\rangle\theta$. Since θ is Σ-well sorted, Σ entails

$$\forall\ \tau(\phi[\langle y_1{:}\tau_1\rangle, \ldots, \langle y_n{:}\tau_n\rangle])$$

which normalized is

$$\forall\ \tau_1(y_1)\wedge\cdots\wedge\tau_n(y_n)\rightarrow\tau(\phi[y_1, \ldots, y_n]) \tag{5.15}$$

For $1\leq i\leq n$ let

$$\psi_i[\langle z_{i,1}{:}\tau'_{i,1}\rangle, \ldots, \langle z_{i,m_i}{:}\tau'_{i,m_i}\rangle] \text{ be } \langle y_i{:}\tau_i\rangle\sigma$$

Since σ is Σ-well sorted, for each $1\leq i\leq n$, Σ entails

$$\forall\ \tau_i(\psi_i)$$

which normalizes as

$$\forall(\tau'_{i,1}(z_{i,1}) \wedge \cdots \wedge \tau'_{i,m_i}(z_{i,m_i}) \rightarrow \tau_i(\psi_i[z_{i,1}, \ldots, z_{i,m_i}])) \qquad (5.16)$$

By combining each sentence of (5.16) with (5.15) we can conclude the Σ entails

$$\forall(\tau'_{1,1}(z_{1,1}) \wedge \cdots \wedge \tau'_{1,m_1}(z_{1,m_1}) \wedge \cdots \wedge \tau'_{n,m_n}(z_{n,m_n}) \rightarrow \tau(\phi[\psi_1, \ldots, \psi_n]))$$

which is the normalization of

$\forall\ \tau(\langle x:\tau\rangle\theta\sigma)$ ■

The closure of the well sorted substitutions under pairwise composition, as stated by the second lemma above, is crucial to the viability of considering only well sorted substitutions. In previous chapters the most common way of obtaining new substitutions is by composing existing substitutions. The closure under composition of the set of well sorted substitutions is necessary if we are to compute with well sorted substitutions in manners akin to the way we compute with ordinary substitutions.

Section 3.2 defines what it means for a substitution to be a unifier, for one substitution to be more general than another, for one set of substitutions to be a complete set of another, and for a set of substitutions to be most general. All of these notions can be adapted to well sorted substitutions as follows:

Definition: Well Sorted Unifier
Let E be a set of expressions and θ be a substitution. θ is a *well sorted unifier* of E relative to Σ if it is a unifier of E and is well sorted relative to Σ.

Definition: Σ–More General
Let θ_1 and θ_2 be substitutions that are well sorted relative to Σ. θ_1 is Σ–*more general* than θ_2 (written $\theta_1 \geq_\Sigma \theta_2$) iff $\theta_1 \cdot \sigma = \theta_2$ for some substitution σ that is well sorted relative to Σ.

Definition: Σ–Complete Set
Let Θ' and Θ be sets of substitutions that are well sorted relative to Σ. Then Θ' is a Σ–*complete set* of Θ iff:

(1) Θ' is correct; if $\theta' \in \Theta'$ and $\theta' \geq_\Sigma \theta$ then $\theta \in \Theta$.
(2) Θ' is complete; if $\theta \in \Theta$ then for some $\theta' \in \Theta', \theta' \geq_\Sigma \theta$.

Definition: Σ–Most General Set of Substitutions
A set of Σ–substitutions is Σ–*most general* if it does not contain two distinct substitutions such that one is Σ–more general than the other.

The Unification Theorem assures us that any finite unifiable set of expression has a singleton *MGCU*. However, this is not the case for Σ–unifiers; for some Σ there exist Σ–unifiable sets of expressions that have non-singleton Σ*MGCU*s (Σ–most general Σ–complete sets of Σ

unifiers). Here is an example:

Let $\Sigma = \{\ \forall x,y\ ODD(x) \wedge ODD(y) \rightarrow EVEN(plus(x,y))\ ,$
$\forall x,y\ EVEN(x) \wedge EVEN(y) \rightarrow EVEN(plus(x,y))\ \}$

Let $E = \{z{:}EVEN,\ plus(v,w)\}$

Then $\theta_1 = \{plus(x{:}EVEN, y{:}EVEN)/z,\ x{:}EVEN/v,\ y{:}EVEN/w\}$

and $\theta_2 = \{plus(x{:}ODD, y{:}ODD)/z,\ x{:}ODD/v,\ y{:}ODD/w\}$

are each Σ-unifiers of E and

$\Theta = \{\theta_1, \theta_2\}$ is a $\Sigma MGCU$ of E.

Notice that neither θ_1 nor θ_2 is Σ-more general than the other. Indeed, a finite set of expressions may have an infinite $\Sigma MGCU$. For instance:

Let $\Sigma = \{\ \forall x T(i(x)) \rightarrow T(i(s(x))),\ T(i(a))\ \}$

Let $E = \{z{:}T,\ i(s(y))\}$

Then $\{\{a/y, i(s(a))/z{:}T\}, \{s(a)/y, i(s(s(a)))/z{:}T\}, \cdots\}$ is a $\Sigma MGCU$ of E.

We now consider the Sorted Unification Algorithm, which given two expressions and a sort theory determines whether the expressions are unifiable with respect to the sort theory, and, if so, returns a Σ-complete set of Σ-unifiers for the two expressions. The form of the algorithm is similar to that of the ordinary Unification Algorithm. Both algorithms repeatedly find a place where the two expressions differ and remove the difference by applying a substitution. Not all differences between two expressions can be removed by a Σ-substitution, but those that can are said to be Σ*-negotiable*.

The notions of difference and negotiability can be captured by saying that two expressions, one containing the subexpression e_1 where the other contains e_2, have the difference $\{e_1, e_2\}$. The difference set of the two expressions contains all the differences that the two expressions have.

Definition: Difference Set

The *difference set* of expressions E and E' (written $DIFF(E,E')$) is defined as:

$DIFF(E,E') = \emptyset$ if E and E' are the same expression.

$= DIFF(e_1,e_1') \cup \cdots \cup DIFF(e_n,e_n')$

if E is composed of constituents $e_1, e_2, \ldots, e_n$ and E' is composed in the same manner of constituents $e_1', e_2', \ldots, e_n'$,

$= \{\{E,E'\}\}$ otherwise.

Definition: Σ-Negotiable

A pair of expressions $\{e_1, e_2\}$ is Σ-negotiable iff at least one of e_1, e_2 is a variable, say $x{:}\tau$, and the other is a term, say t, such that $x{:}\tau$ does not occur in t and $\Sigma \models \exists(\tau(t))$.

The Sorted Unification Algorithm makes non-deterministic choices of the same kind as those made by the GSRA and the SSRA. Each successful execution path through the algorithm results in the output of a single Σ–unifier. The set of all outputs produced by all execution paths is a Σ–complete set of Σ–unifiers. If all execution paths terminate in FAILURE, then the input expressions are not Σ–unifiable.

Sorted Unification Algorithm
Input: expressions A and B of SFOPC, and Σ, a sort theory
Output: SUCCESS or FAILURE; If SUCCESS, then a substitution is also output

```
(1)  let $\sigma = \epsilon$
(2)  while $A\sigma \neq B\sigma$ do
(3)      select $\{U,V\} \in DIFF(A\sigma, B\sigma)$
(4)      if $\{U,V\}$ is negotiable then
(5)          let $x{:}\tau$ be whichever of U,V is a variable and let $t$ be the other
(6)          let $\Theta$ be any Σ–complete set of Σ–substitutions such that
                 for every $\theta \in \Theta$, $\Sigma \models \forall\tau(t\theta)$
(7)          choose $\lambda \in \Theta$
(8)          let $\sigma = \sigma \cdot \lambda \cdot \{t/x{:}\tau\}$
(9)      else FAIL
(10) SUCCEED with output $\sigma$
```

In step (3) a difference is selected from the difference set of $A\sigma$ and $B\sigma$. This is intended to be the same kind of selection as considered in the discussion of selection functions. Recall that selections, unlike choices, do not require different alternatives to be considered.

In step (5), if both U and V are variables then it does not matter which is taken as $x{:}\tau$.

As previously pointed out, it is possible for two expressions to have an infinite $\Sigma MGCU$. In the algorithm this could arise in step (6) where there may not be a finite Σ–complete set of Σ–substitutions Θ such that for every $\theta \in \Theta$, $\Sigma \models \forall\tau(t\theta)$. This is not necessarily a problem if all we want to do is determine whether two expressions are unifiable, or even if we want to compute a fixed number of unifiers, but it is certainly a problem if we want to compute a Σ–complete set of Σ–unifiers.

Even if every possible choice of Θ is a singleton, step (6) may not be effectively computable. There is no decision procedure for determining whether there is a Θ such that $\Sigma \models \forall\tau(t\theta)$ or even whether $\Sigma \models \tau(t)$ for ground t. This is an immediate consequence of the undecidability of FOPC. Observe that Σ's limitation to monadic predicates has no effect on the decidability since Σ may still contain sentences with arbitrary function signs.

Therefore, if the Sorted Unification Algorithm is to effectively compute a finite Σ–complete set of Σ–unifiers then Σ must be such that

> for any sort expression τ, and any term t there is a finite Σ–complete set of Σ–substitutions such that for every $\theta \in \Theta$, $\Sigma \models \forall\tau(t\theta)$ and, furthermore, there is an effective procedure for finding that set.

An important problem, which is not addressed in this thesis, is the identification of a set of syntactic constraints that guarantee that a sort theory has the above property.

5.5. Properties of RT

To begin with, RT is an extension of RQ; that is, over the sentences of FOPC RT agrees with RQ.

RQ–RT Equivalence Theorem
A FOPC sequent is in $\models_{RQ}$ iff it is in $\models_{RT}$.

Proof
if clause: Trivial since every RQ–model is an RT–model.
only-if clause: If $\langle M, S^A \rangle$ is an RT–countermodel to a FOPC sequent then M is an RQ–countermodel to that sequent. ■

Under the Tarskian semantics for FOPC the rule of Universal Instantiation, which derives an instance of a universally-quantified formula, and the rule of Existential Generalization, which derives an existentially-quantified formula from an instance of it, are both sound. The following lemmas state a corresponding result for SFOPC, under both the RT and the T model theories.

Instantiation and Generalization Lemmas
Let M be any RT model, ψ be any A–formula of SFOPC, Σ be any sort theory, and θ be any Σ–substitution. Then:
Instantiation Lemma: If M satisfies $\forall\psi$ and Σ, then M satisfies $\forall(\psi\theta)$.
Generalization Lemma: If M satisfies $\exists(\psi\theta)$ and Σ, then M satisfies $\exists\psi$.

Since these lemmas hold for any RT model they also hold for any T model.

As with other Herbrand theorems, the Sorted Herbrand Theorem pertains to sequents in Skolem Normal Form. Skolem Normal Form for SFOPC sequents is just as it is for FOPC sequents: sentences in the antecedent must be in universal prenex form and those in the consequent must be in existential prenex form.

Before turning to the Sorted Herbrand Theorem, one last concept must be introduced. If $\langle M, A^S \rangle$ is a Herbrand model then A^S is called a Herbrand sort assignment. The Herbrand sort assignments can be ordered in the following way:

$$A^{S_1} \leq A^{S_2} \text{ iff for every sort symbol } S,\ A^{S_1}(S) \subseteq A^{S_2}(S)$$

As will be seen, the importance of sort theories that have a *least* Herbrand sort assignment looms large in this section. [2]

Sorted Herbrand Theorem for RT

Let $\Sigma, akb \Rightarrow q$ be any SFOPC sequent in Skolem Normal Form such that Σ has a least Herbrand sort assignment. Then $\Sigma, akb \models_{RT} q$ iff $akb_{\Sigma gr} \models_{RT} q_{\Sigma gr}$.

Proof

if clause: From the Instantiation and Generalization Lemmas it follows that if RT model M satisfies $\Sigma \bigcup akb$ and falsifies q, then M satisfies $akb_{\Sigma gr}$ and falsifies $q_{\Sigma gr}$.
only-if clause: If there is a countermodel to $akb_{\Sigma gr} \models_{RT} q_{\Sigma gr}$ then there is a Herbrand countermodel, $\langle M, A^S \rangle$, in which A^S is the least Herbrand sort assignment of Σ. I show that $\langle M, A^S \rangle$ is also a countermodel to $\Sigma, (akb^N)_{gr} \models_{RT} (q^N)_{gr}$. I first consider α, an arbitrary sentence in akb and show that $\langle M, A^S \rangle$ satisfies $(\alpha^N)\theta$, an arbitrary ground instance of α^N. $(\alpha^N)\theta$ is of the form $T_1 \wedge \cdots \wedge T_n \rightarrow \beta$. If $\beta \notin akb_{\Sigma gr}$ then $\Sigma \not\models T_1 \wedge \cdots \wedge T_n$. Since A^S is a *least* Herbrand sort assignment, $\langle M, A^S \rangle$ falsifies $T_1 \wedge \cdots \wedge T_n$ and hence satisfies $(\alpha^N)\theta$. On the other hand, if $\beta \in akb_{\Sigma gr}$ then $\langle M, A^S \rangle$ satisfies β (since it satisfies $akb_{\Sigma gr}$) and therefore also satisfies $(\alpha^N)\theta$. I now show that $\langle M, A^S \rangle$ falsifies $(q^N)\theta$, for any ground substitution θ. $(q^N)\theta$ is of the form $T_1 \wedge \cdots \wedge T_n \wedge \beta$. If $\beta \notin q_{\Sigma gr}$ then $\Sigma \not\models T_1 \wedge \cdots \wedge T_n$. Since A^S is a *least* Herbrand assignment, $\langle M, A^S \rangle$ falsifies $T_1 \wedge \cdots \wedge T_n$ and hence falsifies $(q^N)\theta$. On the other hand, if $\beta \in q_{\Sigma gr}$ then $\langle M, A^S \rangle$ falsifies β (since it falsifies $q_{\Sigma gr}$) and therefore falsifies $(a^N)\theta$. ■

What happens if Σ has more than one minimal Herbrand sort assignment? Consider the knowledge base consisting of Σ and akb:

$$\Sigma = \{ BABY(Ralph) \vee DOG(Ralph) \}$$

$$akb = \{ \forall x{:}DOG\ Annoys(x, Alan),\ \forall x{:}BABY\ Annoys(x, Alan) \}$$

Notice that Σ has two minimal Herbrand sort assignments; one that satisfies only "$BABY(Ralph)$" and another that satisfies only "$DOG(Ralph)$." $akb_{\Sigma gr} = \emptyset$ because Σ does not logically imply any atomic sentences. Now, here is the problem: $\Sigma \bigcup akb$ RT-entails $Annoys(Ralph, Alan)$, but $akb_{\Sigma gr}$ does not.

Reiter (1977) noticed that this difficulty arose in his work on deductive databases. His solution was to insist that Σ satisfied a condition called "*τ-completeness*"—that for every sort symbol S and every term t either $\Sigma \models S(t)$ or $\Sigma \models \neg S(t)$. This condition is equivalent to requiring that Σ has a unique Herbrand sort assignment, not merely a unique *minimal* one. Though Reiter found a sufficient condition, it is grossly over-restrictive. What about the condition that Σ must have a least Herbrand sort assignment? Is it also over-restrictive? After

[2] These least Herbrand sort assignments are akin to the least Herbrand models that are central to the theory of logic programming. In that literature, least models are often called *unique minimal models*.

presenting a lemma, the Necessity/Sufficiency Theorem asserts that the condition is necessary. The proof shows that for any Σ having multiple minimal Herbrand sort assignments, an example like the above baby-and-dog one can be constructed.

Least Model Lemma
Let Ψ be a set of universal prenex-form sentences. Then Ψ has a least Herbrand model iff for any finite set of ground atomic formulas A, $\Psi \models_T A$ implies $\Psi \models_T \alpha$, for some $\alpha \in A$.

Proof
if clause: Let P_G be the set of all ground instances of the clausal form of Ψ. Furthermore, let M be the greatest lower bound of all Herbrand models of Ψ. I assume that for any finite set of ground atomic formulas A, $\Psi \models_T A$ implies $\Psi \models_T \alpha$, for some $\alpha \in A$, and I show that M satisfies P_G and therefore Ψ. Let $C = \neg\alpha_1 \vee \cdots \vee \neg\alpha_n \vee \beta_1 \vee \cdots \vee \beta_m$ be an arbitrary clause in P_G. If some model of Ψ satisfies one α_i then so does M and hence M satisfies C. Otherwise every model of Ψ falsifies every α_i and hence $\Psi \models \beta_1 \vee \cdots \vee \beta_m$. By the assumption, there is an i such that $\Psi \models \beta_i$, and M satisfies β_i and therefore M satisfies C.
only if clause: Assume Ψ has a least Herbrand model M and that A is a set of atomic sentences such that $\Psi \models_T A$. Then it must be that M does not falsify A, and hence satisfies some $\alpha \in A$. But since M is a least model, every Herbrand model of Ψ satisfies α. Therefore $\Psi \models_T \alpha$. ■

Necessity/Sufficiency Theorem
Let Σ be a sort theory where each sentence is in universal prenex form. It is both *necessary and sufficient* that Σ has a least Herbrand sort assignment for the following statement to hold:

> For every Skolem Normal Form SFOPC sequent of the form $\Sigma, \boldsymbol{akb} \Rightarrow q$, $\Sigma, \boldsymbol{akb} \models_{RT} q$ iff $\boldsymbol{akb}_{\Sigma gr} \models_{RT} q_{\Sigma gr}$

Proof
Sufficiency: This is equivalent to the Sorted Herbrand Theorem for RT.
Necessity: I assume that Σ does not have a least Herbrand sort assignment and construct an $\boldsymbol{akb}$ and q such that $\Sigma, \boldsymbol{akb} \models_{RT} q$, but $\boldsymbol{akb}_{\Sigma gr} \not\models_{RT} q_{\Sigma gr}$. By the Least Model Lemma, there is a finite set of ground atomic formulas, $A = \{P_1(\vec{t_1}), \ldots, P_n(\vec{t_n})\}$, such that $\Sigma \models A$ and for every $1 \leq i \leq n$, $\Sigma \not\models P_i(\vec{t_i})$. Let $\boldsymbol{akb}$ be the set of sentences $\{\forall x\, P_i Q(x) \mid 1 \leq i \leq n\} \cup \{R(\vec{t_i}) \mid 1 \leq i \leq n\}$ and let q be $\exists x\, Q(x) \wedge R(x)$. Then $\boldsymbol{akb}_{\Sigma gr} = \{R(\vec{t_i} \mid 1 \leq i \leq n\}$ and $q_{\Sigma gr} = q_{gr}$. Therefore $\boldsymbol{akb}_{\Sigma gr} \not\models_{RT} q_{\Sigma gr}$ even though $\Sigma, \boldsymbol{akb} \models_{RT} q$. ■

The final result of this section shows that the retriever specified by RT can retrieve a fact only if it can retrieve that fact from the SKB and a single sentence of the AKB. Thus, RT meets one of the objectives that motivated its definition: it does not sanction any chaining other than with taxonomic information.

Specialized Chaining Theorem

Let Σ be a universal prenex form sort theory that has a least Herbrand sort assignment, and let Q be a set of facts. Then, $\Sigma, akb \models_{RT} Q$ only if for some $\alpha \in akb$ and $q \in Q$, $\Sigma,\alpha \models_{RT} q$.

Proof

if $\Sigma, akb \models_{RT} Q$

then $akb_{\Sigma gr} \models_{RT} Q$	(Sorted Herbrand Theorem)
then $akb_{\Sigma gr} \models_{RQ} Q$	(RQ–RT Equivalence Theorem)
then $a \models_{RQ} q$, for some $a \in akb_{\Sigma gr}$ and $q \in Q$	(Generalized No–Chaining Theorem)
then $a \models_{RT} q$, for some $a \in akb_{\Sigma gr}$ and $q \in Q$	(RQ–RT Equivalence Theorem)

a is a well sorted instance of some sentence $\alpha \in akb$, and hence by the Universal Instantiation Lemma $\Sigma,\alpha \models_{RT} a$. Therefore, $\Sigma,\alpha \models_{RT} q$ for some $\alpha \in akb$. ■

As with the Sorted Herbrand Theorem, the correctness of the Specialized Chaining Theorem requires that Σ has a least Herbrand sort assignment. Otherwise, it is possible to chain together two A–sentences and a consequence of Σ. For example, once again consider the KB consisting of Σ and akb:

$$\Sigma = \{ BABY(Ralph) \vee DOG(Ralph) \}$$

$$akb = \{ \forall x{:}DOG\ Annoys(x,Alan),\ \forall x{:}BABY\ Annoys(x,Alan) \}$$

Observe that $Annoys(Ralph,Alan)$ RT–follows from $\Sigma \bigcup akb$ but not from any of its subsets.

5.6. Computing Retrievability

This section presents and proves correct an algorithm that decides whether an SFOPC sequent is in the RT–entailment relation. The presentation takes place in two subsections. The first develops an algorithm that solves retrieval problems with restricted schematic variables and the second shows how retrieval problems with restricted quantified variables can be reduced to retrieval problems of the first variety.

5.6.1. Answering Sorted Schematic Queries

A *sorted schematic sentence* of QFPC is a sentence of QFPC in which schematic variables with restrictions may take the place of ordinary terms. $\Sigma, akb \Rightarrow q$ is a *sorted schematic sequent of QFPC* iff Σ is a sort theory whose sentences are in universal prenex form, akb is a set of sorted schematic sentences of QFPC and q is a sorted schematic sentence of QFPC. If $\Sigma, akb \Rightarrow q$ is a sorted schematic sentence of QFPC, then q is retrievable from Σ, akb iff for some Σ–substitution σ, $akb_{\Sigma gr} \models_{RP} q\sigma$.

The *Sorted Schematic Sequent Retrieval Algorithm* (SSSRA) decides whether a sorted schematic CNF sequent of QFPC is in this retrievability relation. This algorithm is identical to the Schematic Sequent Retrieval Algorithm with the exception that step (15) computes ΣMGCUs instead of ordinary MGCUs. Thus, when the SSSRA halts it indicates SUCCESS or FAILURE, and if SUCCESS is indicated the algorithm also outputs a Σ-substitution.

The algorithm yields a provability relation called *SSSRA-provability*.

Definition: SSSRA-provable
Let $\Sigma, akb \Rightarrow q$ be a sorted schematic CNF sequent of QFPC. Then, q is *SSSRA-provable* from $\Sigma \bigcup akb$ (written $\Sigma, akb \vdash_{\overline{SSSRA}} q$) with *extracted answer* θ iff there is some sequence of choices for which the Sorted Schematic Sequent Retrieval Algorithm halts with SUCCESS and outputs θ when input $\Sigma, akb \Rightarrow q$.

The next two results establish that the SSSRA lifts the GSRA. First, the Ground Equivalence Theorem states that the SSSRA and the GSRA behave identically when input a ground sequent. Then the Sorted Lifting Theorem states that computations performed by the SSRA on sorted schematic sequents are themselves schematic for computations on ground sequents.

Ground Equivalence Theorem
Let $kb \Rightarrow q$ be a ground CNF sequent of QFPC. Then $kb \vdash_{\overline{GSRA}} q$ iff $kb \vdash_{\overline{SSSRA}} q$. Moreover, the Ground Sequent Retrieval Algorithm and the Sorted Schematic Sequent Retrieval Algorithm implicitly define isomorphic search spaces. They differ only in that every arc in the schematic search space has an additional label of the form "$\theta_i = \epsilon$".

Sorted Lifting Theorem
Let $\Sigma, akb \Rightarrow q$ be a sorted schematic CNF sequent of QFPC and σ be a Σ-ground substitution for $SVARS(q)$. Then $akb_{\Sigma gr} \vdash_{\overline{SSSRA}} q\sigma$ iff for some Σ-substitution $\gamma \geq_{\Sigma} \sigma$, $\Sigma, akb \vdash_{\overline{SSSRA}} q$ with extracted answer γ.

The Ground Equivalence Theorem can be verified by straightforward examination of the GSRA and the SSSRA. The Sorted Lifting Theorem can be proved by an argument produced by systematically modifying the proof of the Lifting Theorem. Simply replace all occurrences of the words "substitution," "unifier" and "MGCU" with the words "Σ-substitution", "Σ-unifier" and "ΣMGCU" respectively. The resulting argument is correct because all properties of substitutions, unifiers, and MGCUs that the original proof relies on are also properties (as established in Section 5.4) of Σ-substitutions, Σ-unifiers, and ΣMGCUs.

Recall that the statement of the SSRA in Section 3.4 is more general than necessary. Step (15) of that algorithm allows for the situation where two expressions have a non-singleton MGCU, in spite of the fact that the Unification Algorithm always returns a singleton. The justification for this overgenerality is now apparent. By allowing for non-singleton MGCUs, the SSRA and the proof of its Lifting Theorem could be transformed trivially into the SSSRA and a proof of its Sorted Lifting Theorem.

This section concludes by establishing the correctness of the SSSRA.

SSSRA Correctness Theorem
Let $\Sigma, akb \Rightarrow q$ be a sorted schematic CNF sequent of QFPC and $\Gamma = \{\gamma \mid \Sigma, akb \vdash_{\overline{SSSRA}} q$ with extracted answer $\gamma\}$. Then Γ is a Σ-complete set of general answers to $\Sigma, akb \Rightarrow q$.

Proof
Let θ be any ground Σ-substitution. Then:
θ is an answer to $\Sigma, akb \Rightarrow q$

iff $akb_{\Sigma gr} \models_{RP} q\theta$	(Definition of Retrievability)
iff $akb_{\Sigma gr} \vdash_{\overline{GSRA}} q\theta$	(GSRA Correctness Theorem)
iff $akb_{\Sigma gr} \vdash_{\overline{SSSRA}} q\theta$	(Ground Equivalence Theorem)
iff $\Sigma, akb \vdash_{\overline{SSSRA}} q$ with some extracted answer $\gamma \geq_{\Sigma} \theta$	(Sorted Lifting Theorem)

Therefore, if γ is an extracted answer then every ground Σ-substitution $\theta \leq_{\Sigma} \gamma$ for $SVARS(q)$ is an answer. So γ is a generalized answer. Going the other way, if θ is an answer then some $\gamma \geq_{\Sigma} \theta$ is an extracted answer. Hence Γ is a complete set of generalized answers. ■

5.6.2. Computing Retrievability of SFOPC Sequents

This subsection shows the following four steps can be used to decide whether an arbitrary sequent of SFOPC is in the RT-entailment relation:

(1) Transform the sequent into Skolem Normal Form.

(2) Transform the matrix of every A-sentence of the sequent into Conjunctive Normal Form.

(3) Remove all quantifiers from every A-sentence of the sequent and replace the restricted quantified variables with restricted schematic variables.

(4) The resulting sequent is a sorted schematic CNF sequent of QFPC. Hand it over to the Sorted Schematic Sequent Retrieval Algorithm.

Here is a transformation that puts any SFOPC sequent into Skolem Normal Form and a theorem stating that the transformation preserves RT-entailment.

Definition: Skolem Normal Transformation

Let $\Sigma, akb \Rightarrow q$ be a prenex form sequent of SFOPC. Its Skolem Normal Transform is computed by the following steps.

(1) Put Σ into prenex form via the usual method for Tarskian FOPC.

(2) While Σ contains a sentence ϕ that contains an existential quantifier do:

Observe that ϕ is of the form

$$\forall x_1 \forall x_2 \cdots \forall x_n \exists y \; \psi[y]$$

for some $n \geq 0$ and prenex-form formula $\psi[y]$. Choose ξ, some n-ary function symbol that does not occur in $\Sigma, akb \Rightarrow q$ and replace ϕ's occurrence in $\Sigma, akb \Rightarrow q$ with

$$\forall x_1 \forall x_2, \ldots, \forall x_n \; \psi[\xi(x_1, \ldots, x_n)]$$

(3) While akb contains a sentence ϕ that contains an existential quantifier do:

Observe that ϕ is of the form

$$\forall x_1{:}\tau_1 \forall x_2{:}\tau_2 \cdots \forall x_n{:}\tau_n \exists y{:}\tau_y \; \psi[y]$$

for some $n \geq 0$ and prenex-form formula $\psi[y]$. Choose ξ, some n-ary function symbol that does not occur in $\Sigma, akb \Rightarrow q$ and replace ϕ's occurrence in $\Sigma, akb \Rightarrow q$ with

$$\forall x_1{:}\tau_1 \forall x_2{:}\tau_2, \ldots, \forall x_n{:}\tau_n \; \psi[\xi(x_1, \ldots, x_n)]$$

and add

$$\forall x_1, \cdots x_n \; \tau_1(x_1) \wedge \tau_2(x_2) \wedge \cdots \wedge \tau_n(x_n) \rightarrow \tau_y(\xi(x_1, \ldots, x_n))$$

to Σ.

(4) While q contains a universal quantifier do:

Observe that q is of the form

$$\exists x_1{:}\tau_1 \exists x_2{:}\tau_2 \cdots \exists x_n{:}\tau_n \forall y{:}\tau_y \; \psi[y]$$

for some $n \geq 0$ and prenex-form formula $\psi[y]$. Choose ξ, some n-ary function symbol that does not occur in $\Sigma, akb \Rightarrow q$ and replace q's occurrence in $\Sigma, akb \Rightarrow q$ with

$$\exists x_1{:}\tau_1 \exists x_2{:}\tau_2 \cdots \exists x_n{:}\tau_n \; \psi[\xi(x_1, \ldots, x_n)]$$

and add

$$\forall x_1, \cdots x_n \; \tau_1(x_1) \wedge \tau_2(x_2) \wedge \cdots \wedge \tau_n(x_n) \rightarrow \tau_y(\xi(x_1, \ldots, x_n))$$

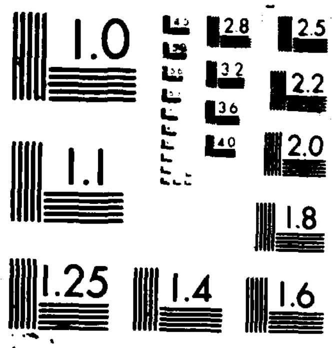
to Σ.

Skolem Normal Transform Theorem

A generalized sequent of SFOPC is in $\vdash_{RT}$ iff its Skolem Normal Transform is.

AD-A189 042 KNOWLEDGE RETRIEVAL AS SPECIALIZED INFERENCE(U) 2/2
ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE A M FRISCH
MAY 87 TR-214 N00014-80-C-0197

UNCLASSIFIED F/G 12/9 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

A prenex form sentence can be placed in CNF by applying the Conjunctive Normal Transformation to its matrix.[3] Since a quantifier-free formula and its Conjunctive Normal Transform are RP-equivalent, they are also RQ- and RT-equivalent. Hence any A-sentence is RT-equivalent to its Conjunctive Normal Transform. Therefore, every A-sentence in an SFOPC sequent can be placed in CNF while preserving the retrievability of the sequent.

After applying these two transformations—the SNT and the CNT— the retrievability of the resulting sequent can be determined by dropping all quantifiers from its A-sentences, replacing its restricted quantified variables by restricted schematic variables, and handing the result over to the SSSRA. The following theorem asserts the correctness of this operation.

SFOPC Retrieval Theorem

Let $\Sigma, akb \Rightarrow q$ be an ordinary SNF sequent of SFOPC and let $\Sigma, akb' \Rightarrow q'$ be the schematic sequent that results from removing all quantifiers from akb and q. Then $\Sigma, akb \models_{RT} q$ iff $\Sigma, akb' \vdash_{SSSRA} q'$.

Proof

$\Sigma, akb \models_{RT} q$

iff $akb_{\Sigma gr} \models_{RT} q_{\Sigma gr}$ (Sorted Herbrand Theorem)

iff $akb_{\Sigma gr} \models_{RQ} q_{\Sigma gr}$ (RQ-RT Equivalence Theorem)

iff for some $qg \in q_{\Sigma gr}$, $akb_{\Sigma gr} \models_{RQ} qg$ (Minuteness Theorem)

iff for some $qg' \in q'_{\Sigma gr}$, $akb'_{\Sigma gr} \models_{RQ} qg'$ (Since $kb_{gr} = kb_{gr}'$ and $q_{gr} = q_{gr}'$)

iff $\Sigma, akb' \vdash_{SSSRA} q'$ (SSSRA Correctness Theorem) ■

[3] See Section 2.5.1.

Chapter 6

Conclusions

The principal contribution of this thesis has been the transformation of retrieval from an ill-defined unstudied process to a formally-defined and well-studied one. The key to the success of this transformation lied in adopting the viewpoint of knowledge retrieval as a specialized inference process. More specifically, knowledge retrieval was argued to be an inference process limited in such a way that it is fundamentally a pattern-matching process, though it may be extended to do a bounded amount of some specialized form of chaining. This characterization of retrieval was formalized by replacing the intuitive notion of pattern matching with the precise notion of no-chaining.

The body of this thesis was devoted to formally specifying a series of four retrievers, culminating in the specification of one that fits the above characterization; it performs all inferences that don't require any chaining, all inferences that involve taxonomic information, and no others.

6.1. External Contributions

The logical developments of this thesis have been motivated by and applied to the study of knowledge retrieval. Nonetheless, they may have important applications outside the study of knowledge retrieval.

In addition to knowledge retrievers, many computational mechanisms used in artificial intelligence manipulate representations. It is my working hypothesis that we can go a long way by specifying and studying these mechanisms as inference engines. This thesis supplies a piece of evidence in support of the hypothesis. The techniques and results of this thesis—especially the model-theoretic specification technique proposed in Chapter 1 and used throughout—may be useful in studying other systems that can be viewed as incomplete inference engines.

RP, the propositional logic presented in Chapter 2, forms the basis of all the retrievers in this thesis. This "logic of no-chaining" is potentially useful as the basis of logics used for other purposes. One example is obvious: RP could form the basis of a logic of explicit belief in the same way that Belnap's four-valued logic forms the basis of Levesque's (1984b) and Lakemeyer's (1986) logics of belief.

Many systems based on automated deduction—deductive databases, theorem provers, logic-programming systems, etc.—are designed to provide exact answers to queries. That is, variables in a query get bound only once in a proof. Indeed, it is crucial that logic programs have this property if they are to be executed by theorem provers that behave at all like interpreters for traditional programming languages. Chapter 4 of this thesis revealed the Strong

Herbrand Property as the semantic counterpart of the exact answer property. The Strong Herbrand Property was shown to be composed of the Herbrand Property and the Minuteness Property. RQ, a model theory possessing the Minuteness Property was constructed from RP, a non-minute model theory. The same construction can be used to construct other minute logics from non-minute ones. Consequently, the analysis of Chapter 4 may have significant and profound application to the theory of systems that compute exact answers.

The part of this thesis with the most scope for application outside the domain of retrieval is the method introduced in Chapter 5 for transforming an unsorted computational logic into a sorted one. Nothing about the method was specific to retrieval per se, and it could be used to transform any logical system that handles quantified variables by using unification. Indeed, this method has been used to add sorts to a logic-programming system (Allen, Giuliano, and Frisch, 1983; Frisch, Allen and Giuliano; 1983) and to the design of a deductive parser (Frisch, 1985b).

There are two reasons why one may want to transform an unsorted logic into a sorted one.[1] One reason is that the additional syntactic devices are useful in building a specification of what entails what. This motivated the introduction of a sorted logic in Chapter 5 where it was used to specify a retriever that performed certain inferences but not others. This, however, provides no motivation for adding sorts to an unsorted Tarskian FOPC since that logic already performs the taxonomic inferences in question. Though the use of a sorted logic may not alter what entails what, it can be used to build efficient deductive mechanisms. Elsewhere (Frisch, 1985b), I have demonstrated that by introducing sorts into a logic smaller deductive search spaces can be built, search spaces that exhibit a minimum-commitment search strategy.

6.2. Extensions

These are some extensions to the current work that are worthy of investigation:

- Develop retrieval algorithms that do not first transform sentences into normal form.
- Redefine RQ and RT so that they handle non-prenex-form sentences. In particular, the new model theories should admit the usual equivalences that allow sentences to be transformed into prenex form.
- Extend the language of SFOPC so that sort atoms can be mixed more freely into S-sentences.
- Endow the retriever with the ability to reason about its own knowledge. Following Levesque's (1984c) approach, this could be done by identifying retrievability with a modal operator in the representation language instead of with a logical-implication relation.
- Extend the retriever to reason about equality.

[1] This argument is general and can be used to motivate other syntactic extensions to a computational logic.

- Specify a retriever that can handle *both* wh-queries and quantifiers.
- Extend SFOPC to allow higher-order restrictions to be placed on variables and build a retriever that can reason about such restrictions. As it now stands, SFOPC allows restrictions to be placed on the values that an individual variable takes on. What I have in mind is placing restrictions on the values that an n-tuple of variables can take on. For example, in the sentence $\forall\langle x,y\rangle{:}{\neq}{>}\; x<y \vee y<x$ the pair of variables must take on pairs of values drawn from the $\neq$ relation. A sort theory then would need to include binary predicate symbols, such as "$\neq$", and perhaps even higher-order predicate symbols. I hypothesize that the techniques developed in Chapter 5 would generalize to this extension in a straightforward manner.

References

Allen, J.F., A.M. Frisch and D.J. Litman, "ARGOT: The Rochester dialogue system," *Proc. of the Second National Conf. on Artificial Intelligence,* August, 1982.

Allen, J.F., M.E. Giuliano and A.M. Frisch, "The HORNE reasoning system," Technical Report 126, Computer Science Dept., Univ. Rochester, December, 1983. Revised September, 1984.

Belnap, N.D., "How a computer should think," in *Contemporary Aspects of Philosophy,* Proc. of the Oxford Int. Symposium, 1975.

Belnap, N.D., "A useful four-valued logic," in G. Epstein and J.M. Dunn (Eds.), *Modern Uses of Multiple-Valued Logic,* Dordrecht: Reidel, 1977.

Bobrow, D.G. and Winograd, T., "An overview of KRL, a knowledge representation language," *Cognitive Science* **1**, 3–46, 1977.

Brachman, R.J., "On the epistemological status of semantic networks," in N.V. Findler (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers,* New York: Academic Press, 1979.

Brachman, R.J., R.E. Fikes and H.J. Levesque, "KRYPTON: A functional approach to knowledge representation," *Computer* **16**, 67–73, October, 1983.

Brachman, R.J., V.P. Gilbert and H.J. Levesque, "An essential hybrid reasoning system: knowledge and symbol level accounts of KRYPTON," *Proc. of the Ninth Int. Joint Conf. on Artificial Intelligence,* August, 1985.

Brachman, R.J., and H.J. Levesque, "The tractability of subsumption in frame-based description languages," *Proc. AAAI-84,* August 1984.

Brown, F.M., "Towards the automation of set theory and its logic," *Artificial Intelligence* **10**, 281-316, 1978.

Cohn, A.G., *Mechanising a Particularly Expressive Many Sorted Logic,* Ph.D. Thesis, Dept. of Computer Science, Univ. Essex, 1983a.

Cohn, A.G., "Improving the expressiveness of many sorted logic," in *Proc. of the Third National Conf. on Artificial Intelligence,* August, 1983b.

Cohn, A.G., "On the solution of Schubert's Steamroller in many-sorted logic," *Proc. of the Ninth Int. Joint Conf. on Artificial Intelligence,* August, 1985.

de Champeaux, D., "A theorem-prover dating a semantic network," *Proc. of the AISB/GI Conf. on Artificial Intelligence,* July, 1978.

Eder, E., "Properties of substitutions and unifications," *Journal of Symbolic Computation* **1**, No. 1, 31-46, 1985.

Fikes, R.E. and N.J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence* **2**, 189–208, 1971.

Frisch, A.M., "Using model theory to specify AI programs," *Proc. of the Ninth Int. Joint Conf. on Artificial Intelligence,* August, 1985a. Also as Research Paper 42, Cognitive Studies Programme, Univ. Sussex, February, 1985. Revised May, 1985.

Frisch, A.M., "Parsing with restricted quantification," Research Paper 46, Cognitive Studies Programme, Univ. Sussex, February, 1985b. Also to appear in *Computational Intelligence,* and in A.G. Cohn and R. Thomas (Eds.), *Recent Advances in Artificial Intelligence,* New York: Wiley, 1986.

Frisch, A.M., "An investigation into inference with restricted quantification and a taxonomic representation," *SIGART Newsletter,* No. 91, January, 1985c. Also in *Proc. of Alvey IKBS Inference Research Theme Workshop 1,* September, 1984. Also as Research Paper 41, Cognitive Studies Programme, Univ. Sussex, September, 1984.

Frisch, A.M. and J.F. Allen, "Knowledge retrieval as limited inference," in D.W. Loveland (Ed.), *Lecture Notes in Computer Science: 6th Conf. on Automated Deduction,* New York: Springer–Verlag, 1982. Also in *Knowledge Representation and Retrieval for Natural Language Processing,* Technical Report No. 104, Computer Science Dept., Univ. Rochester, December 1982.

Frisch, A.M., J.F. Allen and M Giuliano, "An overview of the HORNE logic programming system," *SIGART Newsletter,* No. 84, 1983. Also in *Logic Programming Newsletter,* No. 5, Winter, 1983/4.

Gallaire, H., J. Minker and J.M. Nicolas, "An overview and introduction to logic and databases," in H. Gallaire, J. Minker and J.M. Nicolas (Eds.), *Logic and Data Bases,* New York: Plenum Press, 1978.

Goebel, R., "DLOG: A logic-based data model for the machine representation of knowledge," *SIGART NEWSLETTER,* No. 86, 69–71, October, 1983.

Goebel, R., "Interpreting descriptions in a Prolog-based knowledge representation system," *Proc of the Ninth Int. Joint Conf. on Artificial Intelligence,* August, 1985.

Huet, G. and D.C. Oppen, "Equations and rewrite rules: a survey," Technical Report CSL-111, SRI International, January 1980. Also in R. Book (Ed.), *Formal Languages: Perspectives and Open Problems,* New York: Academic Press, 1980.

Israel, D.J., "On interpreting network formalisms," *International Journal of Computers and Mathematics* **9**, 1–13, 1983a. Also in N. Cercone (Ed.), *Computational Linguistics,* Oxford: Pergamon, 1983a.

Israel, D.J., "On the semantics of taxonomy-based semantic networks," unpublished manuscript, 1983b.

Konolige, K., "Belief and incompleteness," to appear in J. Hobbs and R. Moore (Eds.), *Formal Theories of the Common-Sense World,* Norwood, NJ: Ablex, 1985.

Kowalski, R., *Logic for Problem Solving,* New York: North Holland, 1979.

Kripke, S.A., "Semantical analysis of modal logic I, normal propositional calculi," *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik* **9**, 67–96, 1963.

Lakemeyer, G., "Steps towards a first-order logic of explicit and implicit belief," *Proc. of the Conf. on Theoretical Aspects of Reasoning about Knowledge,* March, 1986.

Levesque, H.J., "A fundamental tradeoff in knowledge representation and reasoning," *Proc. CSCSI/SCEIO 1984,* May, 1984a.

Levesque, H.J., "A logic of implicit and explicit belief," *Proc. AAAI-84,* August 1984. Revised version appears as Technical Report No. 32, Fairchild Laboratory for Artificial Intelligence, August 1984b.

Levesque, H.J., "Foundations of a functional approach to knowledge representation,", *Artificial Intelligence* **23**, 155–212, 1984c.

Lloyd, J.W., *Foundations of Logic Programming,* New York: Springer–Verlag, 1984.

Loveland, D.W., *Automated Theorem Proving: A Logical Basis,* New York: Elsevier North-Holland, 1978.

McSkimin, J.R., and J. Minker, "A predicate calculus based semantic network for deductive searching," in N.V. Findler (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers,* New York: Academic Press, 1979.

Nilsson, N.J., *Principles of Artificial Intelligence*, Palo Alto: Tioga, 1980.

Norman, D.A. and Bobrow, D.G., "On data-limited and resource-limited processes," *Cognitive Psychology* **7**, 44–64, 1975.

Patel-Schneider, P.F., "A decidable first-order logic for knowledge representation," *Proc. of the Ninth Int. Joint Conf. on Artificial Intelligence,* August, 1985. Revised and much-expanded version appears as Technical Report No. 45, Artificial Intelligence Laboratory, Schlumberger Palo Alto Research Center, 1985.

Pearl, J., "On the discovery and generation of certain heuristics," *AI Magazine* **4**, No. 1, 22-33, 1983.

Pigman, V., "The interaction between assertional and terminological knowledge in KRYPTON," *Proc. IEEE Workshop on Principles of Knowledge-Based Systems,* November, 1984.

Plotkin, G.D., "Building-in equational theories," in B. Meltzer and D. Michie (Eds.), *Machine Intelligence* **7**, Edinburgh: Edinburgh University Press, 1972.

Reiter, R., "An approach to deductive question-answering," BBN Technical Report 3649, Bolt Beranek and Newman, Inc., Cambridge, MA., 1977.

Reiter, R., "On closed world data bases," in H. Gallaire and J. Minker (Eds.), *Logic and Data Bases,* New York: Plenum Press, 1978.

Reiter, R., "Equality and domain closure in first order data bases," *Journal of the A.C.M.* **27**, 235-249, 1980.

Reiter, R. "On the integrity of typed first-order data bases," in H. Gallaire, J. Minker and J.M. Nicolas (Eds.), *Advances in Data Base Theory,* Vol. 1, Plenum Press, 1981.

Reiter, R., "Towards a logical reconstruction of relational database theory," in M.L. Brodie, J. Mylopoulos and J.W. Schmidt (Eds.), *On Conceptual Modelling,* New York: Springer-Verlag, 1984.

Robinson, J.A., "A machine-oriented logic based on the resolution principle," *Journal of the A.C.M.* **12**, 23-41, 1965.

Robinson, J.A., *Logic: Form and Function,* New York: North-Holland, 1979.

Robinson, J.A. and Sibert, E.E., *The Loglisp User's Manual,* School of Computer and Information Science, Syracuse Univ., December, 1981.

Sanderson, J.G., "The Lambda Calculus, Lattice Theory and Reflexive Domains," Mathematical Institute Lecture Notes, University of Oxford, 1973.

Shapiro, S.C., "The SNePS semantic network processing system," in N.V. Findler (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers,* New York: Academic Press, 1979.

Stickel, M.E., "Automated deduction by theory resolution," *Journal of Automated Reasoning* **1**, 333-355, 1985.

Tarjan, R.E. and J. van Leeuwen, "Worst-case analysis of set union algorithms," *Journal of the A.C.M.* **31**, 245-281, 1984.

Tarski, A., "Der wahrheitsbegriff in den formalisierten sprachen," *Studia Philosophica* **1**, 261-405, 1935. Translated as "The concept of truth in formalized languages," in A. Tarski (Ed.), *Logic, Semantics, and Mathematics,* Oxford: Clarendon Press, 1956.

Walther, C. "A many-sorted calculus based on resolution and paramodulation," Interner Bericht Nr. 34/82, Universitat Karlsruhe, Fakultat fur Informatik, 1982. Also in *Proc. of the Eighth Int. Joint Conf. on Artificial Intelligence,* August, 1983.

Walther, C., "A mechanical solution of Schubert's Steamroller by many-sorted resolution," *Proc. of the Fourth National Conf. on Artificial Intelligence,* 1984a. Also in *Artificial Intelligence* **26**, 217-224, 1985.

Walther, C., "Unification in many-sorted theories," *Proc. of the Sixth European Conf. on Artificial Intelligence,* 1984b.

Walther, C., "Schubert's Steamroller – a case study in many-sorted resolution," Interner Bericht Nr. 5/84, Universitat Karlsruhe, Fakultat fur Informatik, 1984c.

Warren, D.H.D., "Efficient processing of interactive relational database queries expressed in logic," *Proc. of the Seventh Int. Conf. on Very Large Data Bases,* IEEE, Cannes, France, 1981.

Warren, D.H.D. and F.C.N. Pereira, "An efficient easily adaptable system for interpreting natural language queries," *American Journal of Computational Linguistics* **8**, 110-122, 1982.